Universidad Complutense de Madrid

Facultad de Informática

Proyecto de Sistemas Informáticos

Optimización térmica del banco de registros mediante técnicas de compilación

Autores:

Abel Crespillo Rodríguez
Antonio Gastón Fernández
Celia Lorenzo Lorenzo

Profesores directores:

David Atienza Alonso
José Luis Ayala Rodrigo

Curso:

2009/2010

AUTORIZACIÓN

Los alumnos Abel Crespillo Rodríguez, Antonio Gastón Fernández y Celia Lorenzo Lorenzo autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

En Madrid, a 11 de Junio de 2010

RESUMEN

El avance en las tecnologías de integración sub-micrónica ha permitido la implementación de complejos sistemas electrónicos, Sin embargo, el coste a pagar ha sido un incremento de la densidad de potencia y, por tanto, la temperatura disipada en la superficie del chip. Esta temperatura tiene diversas repercusiones negativas por su relación con la potencia estática y la fiabilidad de los componentes.

El banco de registros de un procesador es uno de los “puntos calientes” más importantes. Su temperatura, ligada al uso de los registros, puede ser optimizada mediante políticas de asignación de registros eficientes.

Este trabajo propone dos políticas de asignación de registros, integrados en un entorno de compilación industrial, que optimizan las principales métricas térmicas.

ABSTRACT

Current state of sub-micron integration technologies has allowed the implementation of complex electronic systems. However, the drawback is an increase in the power and, therefore, in the temperature dissipated on the chip surface. This temperature has several negative impacts, due to its relation with static power consumption and reliability of the devices.

The register file in the processor architecture is one of the most critical “hot spots”. Its temperature, which is related to the assignment of registers, can be optimized by means of efficient compilation techniques.

This work proposes two register assignment policies that have been integrated in an industrial compilation flow. These techniques are able to optimize the main thermal metrics.

PALABRAS CLAVE

Compilador, reasignación de registros, SPARC, banco de registros, CoSy, optimización de temperatura.

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

JUSTIFICACIÓN DEL PROYECTO

El gran avance que actualmente están experimentando las tecnologías ha permitido que se vayan introduciendo procesadores más potentes y sobre los que se puedan ejecutar un mayor número de aplicaciones. Sin embargo, existen diversos **problemas tecnológicos** asociados a dicho avance, como es el calentamiento de sus componentes.

El tamaño de los transistores de los que se compone el microprocesador se ha ido reduciendo de forma gradual, por lo que paulatinamente se ha visto incrementado el número de transistores por chip. La miniaturización extrema de dichos componentes electrónicos está desembocando en la manifestación de fenómenos físicos como la **electromigración,** el transporte de material causado por el movimiento gradual de los iones en un elemento conductor. Debido a la reducción tan acusada en el tamaño de los transistores, este fenómeno es sumamente nocivo en el campo de los microprocesadores ya que produce una baja disipación de calor y el consiguiente aumento de la temperatura.

La densidad de integración del procesador va aumentando anualmente en un orden del 30%. Debido a esta alta densidad de integración se produce un aumento significativo del **consumo de potencia**, lo que implica mayor generación de calor. Dicho aumento de la temperatura en el chip puede afectar seriamente al deterioro del microprocesador y repercutir en el rendimiento y fiabilidad del sistema.

Por ello, es necesario disipar la potencia del procesador, comprobando en qué puntos del mismo se consume una mayor cantidad de energía. Estos puntos son denominados "***hotspots***" (puntos calientes) y son conocidos como la principal causa de los problemas de fiabilidad y el incremento significativo del consumo de energía, con el consiguiente aumento de la temperatura.

Uno de los componentes hardware que está considerado como un *hotspot* es el banco de registros, siendo el elemento del procesador que presenta un mayor incremento de la temperatura. Dicho incremento se produce debido a los **múltiples accesos** que se hacen sobre él en cada ciclo, de manera que los mismos registros físicos sufren una mayor cantidad de accesos. Asimismo, dichos registros normalmente se encuentran próximos en el chip, lo que genera un mayor calentamiento sobre dicha zona del mismo.

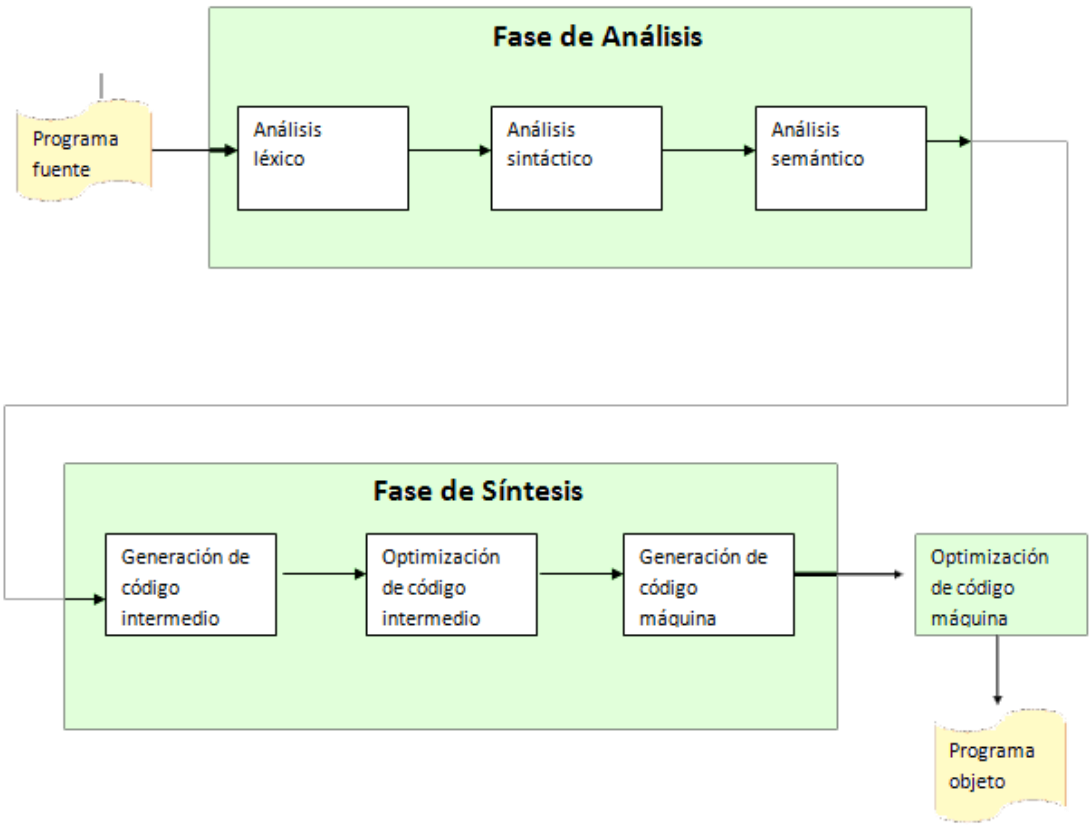
Para resolver dicho problema, se propone modificar un compilador con una metodología basada en la **reasignación de registros**, con la intención de hacer una redistribución de los registros físicos asignados de manera que estén lo más separados posible entre sí y así evitar el excesivo calentamiento del chip en ese punto.

EL COMPILADOR

FLUJO ESTÁNDAR DE COMPILACIÓN

Un compilador es un programa que comprueba secuencialmente que otro programa escrito en lenguaje de alto nivel es gramáticamente correcto y traduce dicho programa a código máquina.

El proceso de compilación se realiza en una serie de fases, que son descritas a continuación:



Esquema general de un compilador

- **Fase de Análisis**

  En esta fase se realiza la comprobación de la corrección del programa fuente, e incluye las fases correspondientes al Análisis Léxico (que consiste en la descomposición del programa fuente en componentes léxicos), Análisis Sintáctico (agrupación de los componentes léxicos en frases gramaticales) y Análisis Semántico (comprobación de la validez semántica de las sentencias aceptadas en la fase de Análisis Sintáctico).

- **Fase de Síntesis**

  El objetivo de esta fase es la generación de la salida expresada en el lenguaje objeto y suele estar formado por una o varias combinaciones de fases de Generación de Código (normalmente se trata de código intermedio o de código

objeto) y de Optimización de Código (en las que se busca obtener un código lo más eficiente posible).

Alternativamente, las fases descritas para las tareas de análisis y síntesis se pueden agrupar en **Front-end** y **Back-end**:

- **Front-end:**

  Es la parte que analiza el código fuente, comprueba su validez, genera el **árbol de derivación** y rellena los valores de la tabla de símbolos. Esta parte suele ser independiente de la plataforma o sistema para el cual se vaya a compilar, y está compuesta por las fases comprendidas entre el Análisis Léxico y la Generación de Código Intermedio.

- **Back-end:**

  Es la parte que genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por el Front End.

Esta división permite que el mismo Back-End se utilice para generar el código máquina de varios lenguajes de programación distintos y que el mismo Front-End, que sirve para analizar el código fuente de un lenguaje de programación concreto, sirva para generar código máquina en varias plataformas distintas. Suele incluir la generación y optimización del código dependiente de la máquina.

El código que genera el Back-End normalmente no se puede ejecutar directamente, sino que necesita ser enlazado por un programa enlazador (linker).

IMPACTO DEL COMPILADOR EN LA TEMPERATURA

Los compiladores juegan un papel muy importante a la hora de llevar un control sobre la **temperatura del sistema.** En la fase de **asignación de registros** que realiza el compilador pueden ocurrir los siguientes problemas:

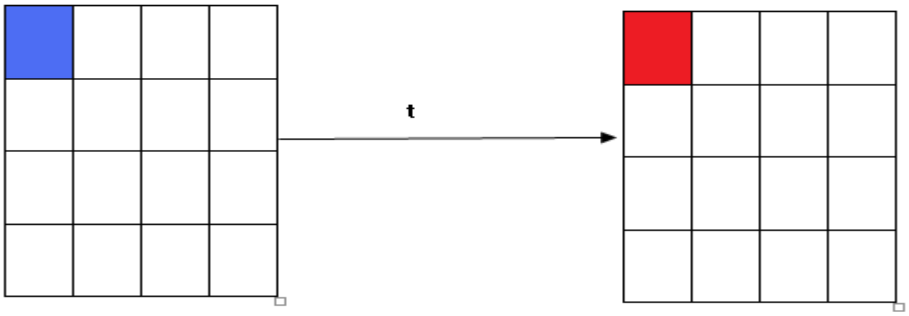
- **Acceso concurrente al mismo registro:**

Cuando tiene lugar la asignación de registros, el compilador siempre esté asignando distintos registros lógicos al mismo registro físico y, por consiguiente, dicho registro físico sufrirá un mayor calentamiento que el resto de registros.

Un ejemplo de programa escrito en pseudocódigo donde el compilador se refleja la situación anteriormente descrita sería:

```
For (int i = 0; i < 100; i++) {
    x = x + 9 + 4 * i;
}
y=x;
For (int j = 0; j < 100; j++) {
    y = y * j;
}
```

La siguiente figura ilustra dicho problema:



- **Acceso a registros localizados en un mismo área:**

El compilador, a la hora de realizar la traducción de diversos programas, se puede encontrar en el caso de que tenga que hacer una asignación de registros físicos de manera que dichos registros se encuentren próximos entre sí dentro del chip. Estos registros son los que sufrirán un mayor aumento de la temperatura. Con el paso

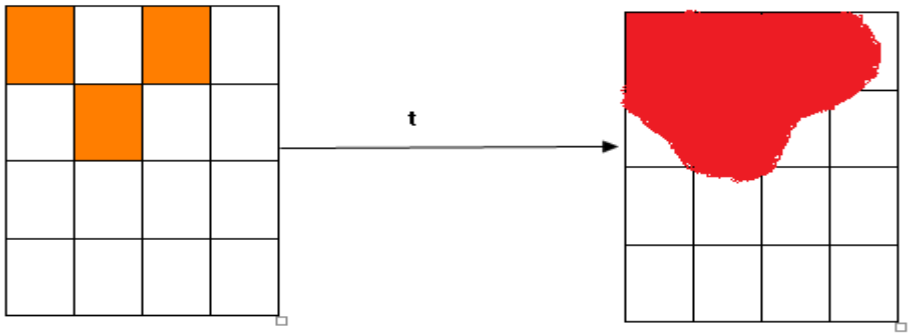
del tiempo, los accesos a los mismos registros provocarán un aumento de temperatura alrededor de la zona donde se encuentran dichos registros.

Esto ocurre debido al fenómeno de la difusión que sufre la temperatura. La transmisión de calor es el flujo de energía térmica y siempre va asociado a una diferencia de temperaturas, pudiéndose producir por conducción, convección y radiación térmica. En este caso se produce una conducción de energía térmica por gradiente de temperatura.

Un ejemplo de pseudocódigo que represente el comportamiento del compilador a la hora de llevar a cabo la asignación de registros tal y como ha sido descrito anteriormente sería:

```
For (int i = 0; i < 100; i++) {
    R1 = 9 + 4 * i;
    R3 = 8 + R1;
    R6 = 3 * i + R3;
}
```

Gráficamente, se puede representar dicha situación con la siguiente figura y ver el impacto térmico que se produce en el tiempo:



Es necesario realizar un análisis para optimizar el comportamiento térmico del banco de registros. La temperatura alcanzada por cada registro en el banco de registros es un factor que depende del **número de accesos** al registro en cuestión, de la

temperatura alcanzada por los registros vecinos, y por la transferencia térmica promovida por las unidades funcionales más calientes cerca del banco de registros. Todos estos parámetros son considerados por los diferentes modelos térmicos para caracterizar el perfil de temperatura.

Se puede conseguir una optimización de la temperatura dentro del banco de registros realizando una reasignación de los registros físicos en tiempo de compilación, logrando dicha optimización sin gastos extra en el rendimiento ni en el hardware. Con estos métodos se consigue una **disminución de temperatura**, ya que los registros físicos a los que se accede más frecuentemente se encuentran más separados en el chip tras dicha reasignación.

Para realizar dicha separación se pueden tener en cuenta varios factores. En primer lugar, influye en gran medida la **distancia** existente entre dos registros físicos asignados por el compilador: cuanta mayor separación exista entre cada par de registros asignados, menor disipación de calor se producirá entre dichos componentes físicos al no estar en contacto. Por otra parte, también influye notablemente la distancia de cada registro al **borde del chip**. Los registros más externos se benefician de una mayor disipación de calor, ya que dichos registros estarán en contacto con un área menor de chip de silicio. Finalmente, y no menos importante, el **tiempo de vida** de cada variable asignada a cada registro lógico determina el intervalo de tiempo por el cual el registro físico asignado a dicho registro lógico está siendo accedido a lo largo de la ejecución del programa, por lo que conviene alejar lo máximo posible registros con grandes tiempos de vida.

OBJETIVOS

Los objetivos que se han pretendido alcanzar mediante la realización de este proyecto de Sistemas Informáticos han girado en torno a la optimización de la temperatura en el banco de registros mediante la **reasignación de registros** físicos en tiempo de compilación.

Hemos realizado un estudio de cómo distribuir la temperatura de una manera más uniforme dentro del banco de registros. Dicho estudio se ha realizado sobre la arquitectura de un **procesador real**. Concretamente, la arquitectura escogida ha sido SPARC-V9. Posteriormente se darán algunas reseñas sobre dicha arquitectura para que el lector conozca un poco más en profundidad sus características.

Tras la realización de dicho estudio, hemos realizado el diseño y la consiguiente **implementación** de múltiples algoritmos de asignación de registros que consiguiesen optimizar la temperatura generada en tiempo de ejecución de los programas, lo cual ha sido en todo momento nuestro objetivo principal.

Un objetivo derivado del anterior es la **integración** de dichos algoritmos en flujo de compilación. Para ello, hemos optado por utilizar un compilador industrial de alta gama como es el sistema de desarrollo de compiladores CoSy. De igual manera, posteriormente se darán ciertas reseñas para que el lector conozca sus características.

Otro de los objetivos derivados ha sido la **verificación** de nuestros algoritmos implementados mediante bancos de prueba o *benchmarks,* programas realistas que proporcionan una carga real de trabajo al sistema. Dichos bancos de prueba han sido compilados en CoSy una vez realizadas nuestras modificaciones sobre él, obteniendo los resultados que posteriormente mostraremos y describiremos.

2. ESTADO DEL ARTE

Los puntos donde se alcanza un mayor nivel de temperatura debido a un mayor consumo de potencia dentro del chip, son los mayores responsables de causar graves problemas de fiabilidad. El banco de registros es conocido como el componente hardware que presenta el mayor calentamiento con respecto al resto de componentes del microprocesador.

Por todo ello, en los últimos años se ha tenido un gran interés por hacer una distribución más eficiente de la temperatura en dicho componente hardware **[1,2]**.

Algunos autores que han trabajado en este tema han presentado diversos y detallados modelos térmicos que se podían resolver eficientemente para realizar una distribución de la temperatura a través del chip. Sin embargo, los resultados obtenidos por dichos modelos presentaban problemas de inexactitud **[3,4]**. Aunque dichos trabajos constituyen un método de análisis para el estudio de la distribución térmica, requieren de un conocimiento completo del diseño. Además, dichos modelos son sólo válidos para un tipo concreto de procesador, ya que no se pueden extender a otras arquitecturas. También se han dado otros enfoques distintos basándose en las medidas dinámicas para caracterizar el comportamiento del chip.

Otros autores han presentado modelos cuyas técnicas permiten caracterizar el comportamiento térmico del chip **[5]**, requiriendo del conocimiento completo de la arquitectura destino así como de las capacidades para modificarlo. Existen otros enfoques que proponen técnicas basadas en las medidas eléctricas para desarrollar un modelo basado en el consumo de energía **[6,7]**.

Todos estos estudios y trabajos realizados han presentado técnicas que en sí no se ocupan del comportamiento térmico del chip. En cambio autores como D. Atienza o J.L. Ayala han realizado estudios y presentado modelos que analizan cómo afecta la

temperatura sobre dicho componente. Además, se han dedicado a la modelización de sistemas térmicos **[8,9]**.

Los trabajos presentados por estos autores son opuestos a los anteriormente mencionados, ya que con sus técnicas no es necesario conocer en profundidad la arquitectura destino para realizar un estudio del comportamiento térmico del chip. Se han basado en el modelo térmico presentado en **[10,11]**, dando un enfoque diferente basado en la simulación que aumenta la granularidad del banco de registros y estudia varios factores que repercuten en el comportamiento de la temperatura.

Existen además estudios realizados sobre tipos de arquitecturas que soportan la ejecución de un mayor número de aplicaciones multimedia, y aunque ofrecen un menor consumo de potencia en comparación con otros procesadores de propósito general (VLIW), existe dicha disipación de energía y por consiguiente un aumento de la temperatura en el chip. En estos casos se han realizado trabajos para disminuir la temperatura no sólo en el banco de registros sino también en el subsistema de memoria, con el fin de optimizar el ancho de banda de dicho sistema **[12]**.

Desde el punto de vista del software se han propuesto varios enfoques para optimizar y reducir el calentamiento del banco de registros. Autores como J. L. Ayala (2007) han presentado diferentes técnicas de compilación para reducir el consumo de energía en dicho componente. Pero estas técnicas de compilación no están dirigidas a sistemas multiprocesador, puesto que con los compiladores actuales no es posible explotar la programación que se encuentra fuera del procesador y el paralelismo a nivel de instrucción que se alcanza no es tan bueno como se podría esperar.

Sin embargo, para sistemas VLIW se han realizado estudios sobre el desenrollado de bucles, aunque se ha demostrado que sigue existiendo un aumento de energía en el banco de registros. El único efecto positivo que posee esta técnica es el de reducir el tiempo de ejecución de los programas. No hay investigaciones previas que estudien el incremento del consumo de energía

realizando desenrollado de bucles, debido a que existen registros que no se utilizan **[13]**.

En el futuro, los dispositivos integrados tendrán que ejecutar aplicaciones multimedia que van a requerir una enorme complejidad computacional. El banco de registros es uno de los principales componentes que va a seguir sufriendo un mayor consumo de energía, por lo que una gestión inadecuada de sus recursos puede repercutir seriamente en el rendimiento y fiabilidad del sistema **[14]**.

También en los últimos años se ha realizado un intenso trabajo a nivel del compilador en la programación de los procesadores VLIW. Muchos de estos trabajos proponen apagar ciertas unidades para realizar un ahorro de energía. Algunos de ellos **[15]** proponen el ahorro de energía en el banco de registros. Sin embargo, estos enfoques no consideran la temperatura como métrica y no se realiza por tanto una optimización de la misma.

En Narayanan **[16]**, se han propuesto varias técnicas para minimizar la temperatura reduciendo la potencia mediante el compilador.

Además, Donad y Martonosi **[17]** han propuesto varias técnicas para gestionar la temperatura en arquitecturas multicore. En ellas se investiga la reducción de la temperatura en el banco de registros.

Otro trabajo reciente, es el realizado por Zhou **[18]**, que propone un algoritmo de reasignación de registros para minimizar la densidad de energía en el banco de registros. Sin embargo este estudio sólo sirve para unas pocas arquitecturas VLIW. Dichas arquitecturas VLIW también se han examinado en **[19]** donde se propone un algoritmo para minimizar la temperatura.

Autores como J. L. Ayala, M. Sabry y D. Atienza proponen trabajos que difieren de los anteriores enfoques estáticos. Dichos autores han realizado trabajos que se basan en la minimización de varios parámetros térmicos relacionados con la fiabilidad, así como

la temperatura media y máxima del banco de registros, así como también el porcentaje de puntos de acceso **[20]**.

El trabajo que proponemos nosotros se basa en la optimización de la temperatura en el banco de registros mediante la **reasignación de registros** físicos en tiempo de compilación. Dicha optimización consigue distribuir la temperatura de una manera más uniforme dentro del banco de registros. Aunque, como hemos visto anteriormente, se han propuesto algoritmos de reasignación de registros para minimizar la temperatura en el chip, todos ellos han sido pensados para arquitecturas VLIW.

Por el contrario, nuestro estudio se ha realizado sobre la arquitectura de un **procesador real** SPARC-V9. Este hecho nos ha permitido verificar la funcionalidad de los algoritmos propuestos en dicho procesador, siendo éste un punto novedoso respecto a trabajos anteriores. Otra ventaja de nuestro trabajo es que los algoritmos realizados son escalables a diversas arquitecturas de procesadores, lo que le otorga a nuestro estudio un amplio abanico de áreas de aplicación.

Todo ello ha sido posible gracias a que nos hemos servido del sistema de desarrollo de compiladores CoSy, ya que su estructura modular ofrece una mayor **facilidad de adaptación** a distintas arquitecturas destino. De esta forma, hemos podido realizar pruebas sobre nuestros algoritmos para verificar su correcto funcionamiento mediante benchmarks, que son programas realistas que proporcionan una carga real de trabajo al sistema, mientras que casi todas las propuestas anteriores se quedan en el apartado teórico pero no muestran resultados prácticos y realistas.

3. CONOCIMIENTO BÁSICO

ARQUITECTURA SPARC

SPARC es una arquitectura RISC big-endian, es decir, con un conjunto reducido de instrucciones, y cuyo orden de escritura en memoria se hace desde los índices más bajos hacia los índices más altos. Ha sido la primera arquitectura RISC abierta cuyas especificaciones de diseño han sido publicadas.

Una de las ideas innovadoras de esta arquitectura es la ventana de registros, que permite hacer fácilmente compiladores de alto rendimiento y una significativa reducción de memoria en las instrucciones load/store en relación con otras arquitecturas RISC. Esta arquitectura cuenta con una serie de ventanas de registros solapadas, en las que cada ventana está formada por 32 registros de punto fijo. De dichos registros, 8 son registros globales compartidos por todas las ventanas, y los 24 restantes son particulares para cada ventana. Dentro de este conjunto de registros se encuentran el puntero a la pila de memoria y el registro apuntador de marco, que se usa para direccionar variables en el marco de pila.

La arquitectura SPARC cuenta con 32 registros enteros de 32 bits y 16 registros de punto flotante de 64 bits (para el caso de doble precisión) que se pueden utilizar como 32 registros de 32 bits (para precisión simple). La siguiente figura muestra la ubicación de los registros de propósito general en cada ventana:

i7	r[31]
i6	r[30]
i5	r[29]
i4	r[28]
i3	r[27]
i2	r[26]
i1	r[25]
i0	r[24]
l7	r[23]
l6	r[22]
l5	r[21]
l4	r[20]
l3	r[19]
l2	r[18]
l1	r[17]
l0	r[16]
o7	r[15]
o6	r[14]
o5	r[13]
o4	r[12]
o3	r[11]
o2	r[10]
o1	r[9]
o0	r[8]
g7	r[7]
g6	r[6]
g5	r[5]
g4	r[4]
g3	r[3]
g2	r[2]
g1	r[1]
g0	r[0]

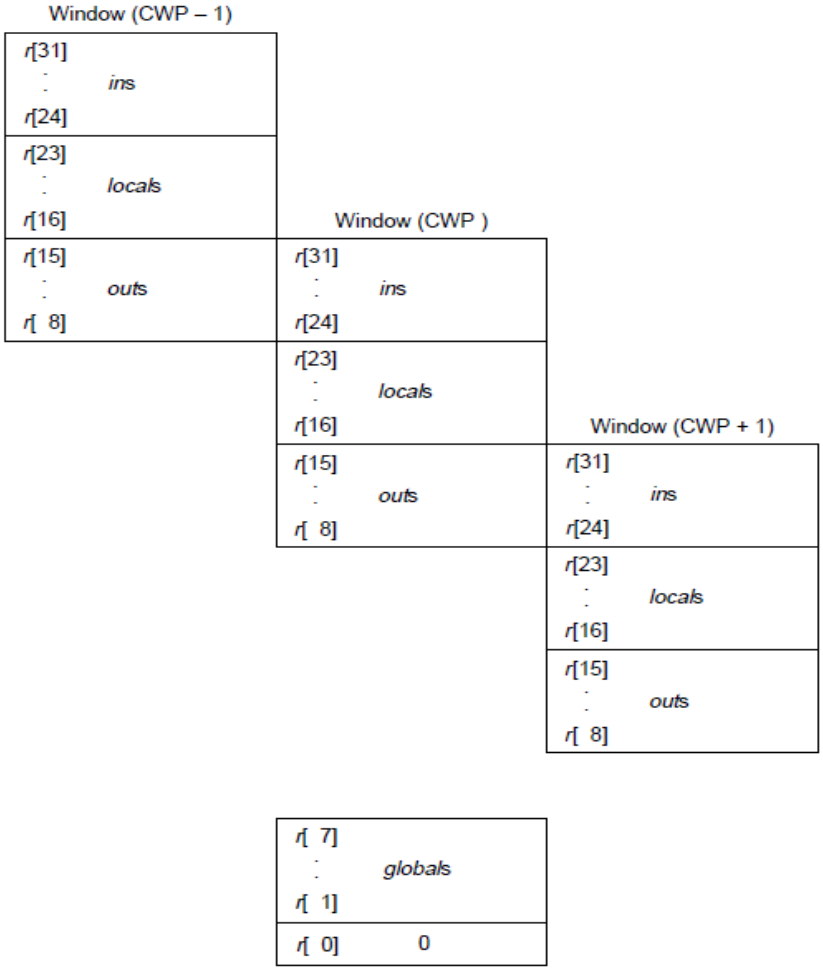
Como ya se ha mencionado anteriormente, la ventana con solapamiento de registros es un rasgo único que caracteriza al diseño de la arquitectura SPARC. El procesador posee muchos más que 32 registros enteros, pero en cada instante únicamente son visibles 32 de ellos:

- De r0 a r7, Registros GLOBALES.
- De r8 a r15, Registros de SALIDA.
- De r16 a r23, Registros LOCALES.

- De r24 a r31, Registros de ENTRADA.

Los registros globales son "vistos" (compartidos) por todas las ventanas, los locales son solo accesibles por la ventana actual y los registros de salida se solapan con los registros de entrada de la ventana siguiente (los registros de salida para una ventana deben ponerse como registros de entrada para la próxima, y deben estar en el mismo registro). El número de ventanas con solapamiento de registros puede oscilar entre 3 y 32.

La siguiente figura ilustra perfectamente dicho solapamiento:



Los registros son asignados directamente por el compilador en la fase de asignación de registros. Hay registros que no pueden ser reasignados, se trata de los 8 registros globales, del puntero de pila y del registro apuntador de marco. Dichos registros **no pueden ser reasignados** por ser registros reservados a la hora de compilar los programas.

COSY COMPILER DEVELOPMENT SYSTEM

CoSy es un sistema de desarrollo de compiladores muy flexible, basado en un diseño modular, cuya estructura se corresponde con la siguiente imagen:

En dicha estructura distinguimos dos partes diferenciadas: la Representación Intermedia (IR) y los motores conectados a ella. Primeramente, la Representación Intermedia en CoSy es una representación estandarizada del código fuente del programa con el que el compilador construye un código objeto. Cada uno de los motores conectados a la IR es una unidad funcional que crea, modifica o traslada dicha representación intermedia del código fuente del programa. Cada motor opera independientemente, sin requerir información explicita sobre los demás motores, devolviendo una nueva IR.

Las tareas del compilador CoSy son:

- Reorientar el motor de back-end (generador de código back-end) proporcionando una nueva descripción de la arquitectura destino.

- Conectar (plug and play) con los motores existentes (sistemas de interacción), ajustar la configuración y las opciones.

- Añadir nuevos motores para la optimización de aplicaciones o arquitecturas específicas.

La modularidad de Cosy permite que estas tareas sean realizadas en paralelo.

La IR se puede dividir en dos partes: CCMIR (Common CoSy Medium-level Intermediate Representation) y LIR (Low-level Intermediate Representation).

En primer lugar, el CCMIR es una descripción estructural del programa para que todos los códigos fuente se traduzcan, independientemente del lenguaje del código fuente o de la arquitectura de destino.

La especificación del CCMIR está escrita en fSDL (full Structure Definition Language). El fSDL es un lenguaje de alto nivel para describir estructuras de datos únicamente, usando notaciones para dominios, operadores y campos. Este lenguaje puede ser extendido en cada motor añadiéndole funcionalidad genérica o no estándar.

Todos los compiladores CoSy front-end generan código intermedio en formato CCMIR, representando el código fuente del programa como una colección de tipos, objetos, sentencias, expresiones y valores, que se corresponden a los nodos del grafo que define la estructura del CCMIR. En cada nodo se incluyen atributos para detallar la semántica del programa. Las referencias contextuales en el programa fuente son resueltas y representadas por referencias a otros nodos.

La otra parte del IR es el LIR, que es el segundo nivel de la descripción de la estructura generado desde el CCMIR a fin de tener en cuenta las características de destino, por ejemplo la arquitectura y su conjunto de instrucciones. Dicha representación convierte las sentencias CCMIR en pseudo-código que se mapea sobre el conjunto de instrucciones de la arquitectura de destino. Generalmente es utilizado por los motores back-end para procesos de planificación, asignación de registros y empaquetado de instrucciones.

Una vez descrito el IR, es conveniente explicar más detalladamente los motores del CoSy que se conectan a dicho módulo central. Podemos clasificar los motores en varios tipos. En primer lugar, están los motores que crean el CCMIR. Por otro lado, hay algunos motores que analizan el CCMIR. También hay optimizadores de código, que transforman y mejoran el CCMIR, a menudo basándose en el análisis de los motores anteriormente mencionados. Además, existen traductores de código de un lenguaje a otro, por ejemplo de C a CCMIR o de CCMIR a LIR. Por último, hay motores de apoyo para ayudar a la depuración. Por ejemplo, el motor CheckMIR, que controla que el CCMIR sea consistente, y el motor ndump que muestra el CCMIR.

En concreto, los motores que nos interesan son los de asignación de registros y más específicamente el de generación de código fuente. Concretamente nuestro trabajo ha consistido en modificar este último motor, para que una vez obtenidos los resultados del módulo de asignación de registros se generase una reasignación de dichos registros antes de emitir el código final.

Finalmente, es el Supervisor el que construye el EDL (Engine Description Language) para el Generador de Código, que es el motor específico que traduce el IR a código objeto, usando la información contenida en el archivo CGD (Code Generator Description). En dicho archivo se describe el traductor de CCMIR a LIR, el planificador y el asignador de registros, especificando cada uno de los registros.

El planificador, a su vez, interpreta cada regla del LIR como una instrucción. Toma una lista de instrucciones y las reordena en un esquema óptimo, tomando en cuenta dependencias en memoria, latencias y ocupación del procesador. Se puede ejecutar antes, actuando sobre los pseudo-registros, o después de la asignación de registros, utilizando ya los registros físicos que tiene la arquitectura destino.

La principal ventaja del sistema de desarrollo de compiladores CoSy es su facilidad para adaptarlo a distintas arquitecturas destino, gracias a que su estructura modular permite modificar cada motor por separado, sin tener en cuenta el resto. En nuestro caso, hemos utilizado el compilador y generador de código CoSy para la arquitectura SPARC.

LA ASIGNACIÓN DE REGISTROS

Dentro de las etapas de la compilación, la asignación de registros es quizás la fase que más influirá en el rendimiento posterior del sistema, ya que definirá los recursos que el programa compilado requerirá para su correcto funcionamiento. Por ello, es imprescindible que dicha asignación se optimice con el fin de obtener mayor fiabilidad con un menor consumo de recursos (es decir, de registros del banco de registros).

Cada programa está formado por una serie de variables. Dichas variables son llamadas registros lógicos o pseudoregistros, y han sido computadas en el grafo de flujo del programa en fases anteriores de compilación. La tarea del compilador es minimizar el número de solapamientos de las variables temporales necesarias para evaluar las expresiones, con el fin de hacer uso de un menor número de registros físicos simultáneamente. Esta primera fase de la asignación de registros se llama **register allocation**.

El **tiempo de vida** de una variable es el período de tiempo que tiene lugar entre el primer acceso a una variable (primera escritura en el registro) y el último de los accesos a dicha variable

(última lectura de dicha variable). Simultáneamente, varias variables pueden “estar vivas” en un punto del programa. En ese caso, se dice que dichas variables **solapan** en el tiempo. Es evidente ver que variables que solapan en el tiempo no pueden compartir el uso de un mismo registro físico, por lo que el número total de registros físicos que utilizará un programa vendrá dado por el número máximo de variables solapadas simultáneamente a lo largo de la ejecución del programa.

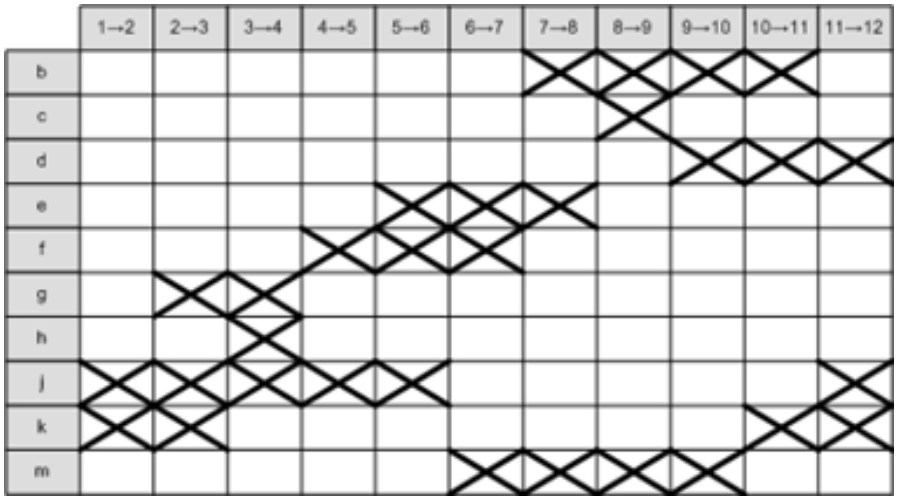


Imagen sacada de http://www.caerolus.com/informatica/teorema-4-colores-aplicado-compiladores.html

Por ello, la primera de las fases que tienen lugar en la asignación de registros consiste en la minimización del solapamiento de los registros lógicos utilizados por el programa. Para ello, la técnica utilizada es el **graph coloring** o coloreado de grafos. Dicha técnica consiste en construir un grafo, llamado grafo de interferencias, en el que cada nodo es una variable lógica del programa. Dos nodos estarán unidos por una arista si dichos nodos interfieren temporalmente.

Tras la construcción de dicha estructura, el problema se reduce a aplicar el algoritmo de coloreado de grafos. Cada color utilizado en el algoritmo representará el uso de un registro físico diferente, por lo que el número total de registros físicos utilizados por dicho programa vendrá dado por el número total de colores utilizados en el algoritmo anteriormente descrito.

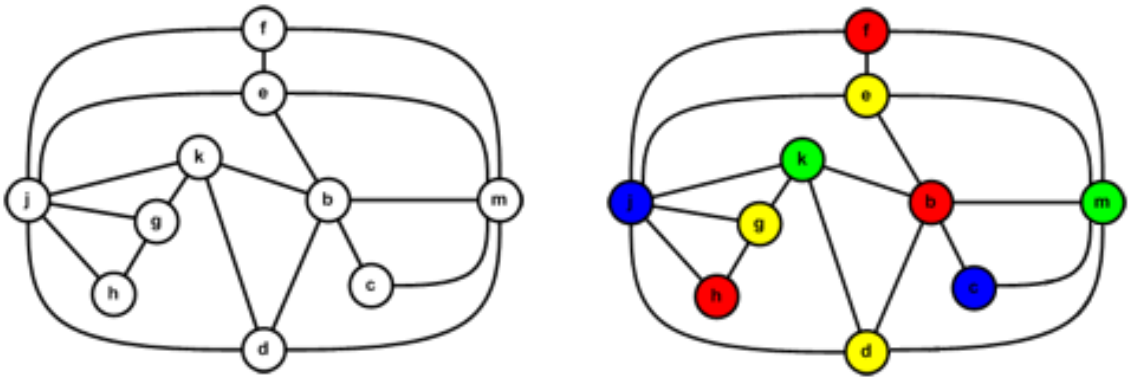


Imagen sacada de http://www.caerolus.com/informatica/teorema-4-colores-aplicado-compiladores.html

Una vez que dicho número de registros ha sido computado, comienza la fase de **register assignment** propiamente dicha. En esta fase se lleva a cabo la asignación de los registros físicos disponibles a las variables temporales del programa. Dicha asignación vendrá dada por los colores utilizados previamente en la fase del coloreado del grafo, y seguirá una política definida en el compilador.

Generalmente, dichas políticas no tienen en cuenta la disipación de calor ni la temperatura alcanzada en el banco de registros, por lo que simplemente asignan el primer registro físico disponible a la variable. Esto implicará que todos los registros físicos que serán asignados por el programa estarán situados próximamente en el banco de registros. Por ello, nuestra labor será reasignar dichos registros con el fin de separarlos lo máximo posible y de manera uniforme a lo largo del área de silicio, consiguiendo menores picos de temperatura y más espaciados.

4. ASIGNACIÓN ESTÁTICA

OBJETIVO DEL ALGORITMO

El primero de los algoritmos que vamos a proponer en este proyecto de Sistemas Informáticos va a consistir en una **asignación estática** de los registros escogidos. Por asignación estática entendemos la que se lleva a cabo sin tener apenas en cuenta tiempos de vida de las variables del programa. Es decir, para lo único que dichos tiempos se tienen en cuenta es para determinar el orden de asignación. Posteriormente desarrollaremos otro algoritmo en el que los tiempos de vida de cada variable serán el eje sobre el que gire la reasignación.

Previamente a introducirnos de lleno en el algoritmo conviene conocer en líneas generales en qué consiste, con el fin de que en todo momento se tenga claro el objetivo principal que se pretende alcanzar. Este algoritmo va a contar el número total de registros físicos que han sido utilizados en la asignación inicial que ha tenido lugar en el ***register assignment*.** Una vez conocido dicho número, se va a buscar cuál es la colocación óptima de dicho número de registros, con el fin de minimizar el calor que su uso va a generar en el chip de silicio.

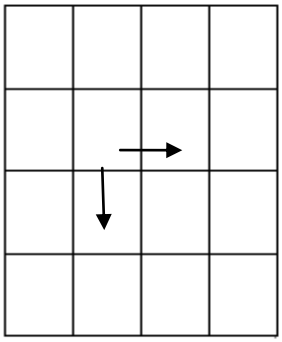
Este algoritmo ha sido planteado como un problema de **optimización multiobjetivo**. Dichos algoritmos buscan la solución óptima teniendo en cuenta una serie de objetivos que tienen que cumplirse para que dicha solución sea la más óptima de todas las disponibles. En este caso, los objetivos que han debido de cumplirse en todo momento han sido:

- La distancia entre cada par de registros escogido ha de ser la máxima posible.
- La distancia de cada registro escogido al borde del chip de silicio ha de ser la mínima posible.

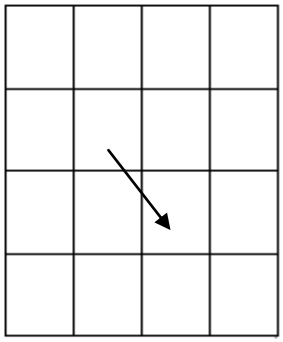
La definición de unos objetivos tan claros ha desembocado en la creación de unas métricas bien definidas. Las métricas han servido para escoger los mejores registros a utilizar. Además, cada uno de los objetivos va a tener un peso específico en la métrica. Posteriormente ampliaremos la información al respecto.

Vamos a explicar cómo se han definido cada uno de los valores anteriores, con el fin de que quede lo suficientemente claro y conciso:

La **distancia entre cada par de registros** se ha definido como el número mínimo de registros físicos en el banco de registros que hay entre dicho par de registros. Más concretamente, horizontal y verticalmente entre cada par de registros hay un registro de separación, y diagonalmente dicha distancia es de dos registros. Hemos tenido en cuenta dicha diferenciación ya que físicamente es mayor el área de silicio que dos registros comparten si están contiguos en horizontal o en vertical que si lo están en diagonal. Veamos dos ejemplos claros de distancias:

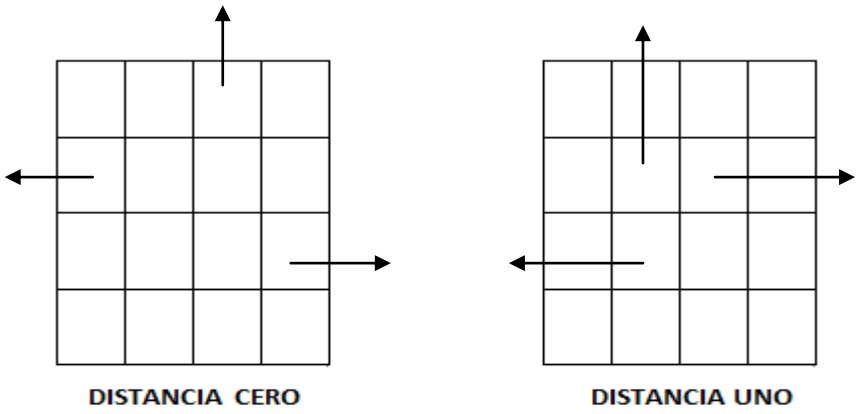


DISTANCIA UNO



DISTANCIA DOS

La **distancia de cada registro al borde** del banco de registros se ha definido como el número mínimo de registros físicos en el banco de registros que hay entre dicho registro y el borde. Veamos varios ejemplos:



En todo momento cada registro poseerá un valor de **métrica**. Dicho valor vendrá dado por una bonificación y por una penalización, cada uno con su correspondiente peso. La bonificación será la distancia más corta con el resto de registros físicos escogidos hasta dicho momento, mientras que la penalización vendrá dada por la distancia de dicho registro al borde del chip.

En cambio, el **peso** se podría definir como el grado de importancia que dicho valor posee en todo el conjunto. El peso puede ser equitativo (mismo peso para bonificación y penalización), favorable a bonificación o favorable a la penalización. Para estudiar cómo el peso otorgado a cada valor influía en el resultado final, hemos variado el valor de cada peso y hemos obtenido resultados claramente diferentes. Dichos resultados serán expuestos más adelante.

$$Metrica_{(x,y)} = (PesoDM * DMin_{(x,y)}) - (PesoDB * DBorde_{(x,y)})$$

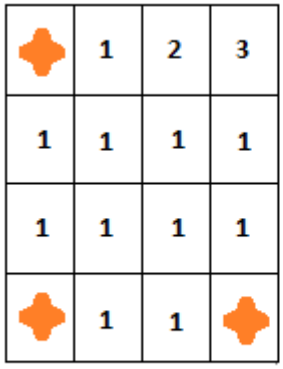
donde PesoDM → peso otorgado a la distancia mínima del registro con el resto
$DMin_{(x,y)}$ → distancia mínima del registro (x,y) con el resto
PesoDB → peso otorgado a la distancia al borde del chip del registro
$DBorde_{(x,y)}$ → distancia al borde del chip del registro (x,y)

Es imposible conocer sobre qué registro se tiene que comenzar asignando (al que llamaremos registro inicial) para conseguir la configuración más eficiente. Por ello, tenemos que probar todos y cada uno de los posibles registros iniciales y evolucionar las métricas del banco de registros para todos los registros que haya que asignar. De esta forma nos aseguraremos que se van a probar todas las configuraciones posibles, y de que

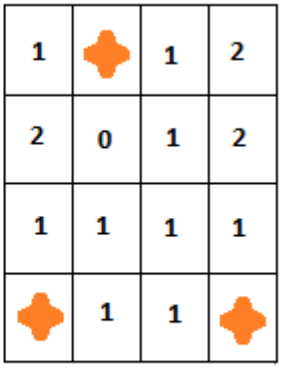
nuestro algoritmo va a conseguir la mejor entre todas las disponibles.

Sin embargo, puede que en la arquitectura sobre la que se pretenda ejecutar este algoritmo existan restricciones respecto de los registros que pueden o no ser reasignados. Tal es el caso de SPARC ya que, como anteriormente se explicó, existen una serie de registros (a los que llamaremos prohibidos) que no pueden ser reasignados ya que están reservado por diversas razones (puntero de pila, etc.). Dichos registros se tienen en cuenta en nuestro algoritmo.

Para saber qué registro físico es el más adecuado en todo momento para escoger, se atenderá a la metrica de cada registro: el que actualmente tenga mayor métrica será escogido como el mejor disponible. Tras escoger todos los necesarios y teniendo las métricas finales actualizadas para cada registro inicial, el algoritmo calculará la suma de todas las métricas y escogerá la configuración con mayor suma total. Es decir, entre dos de las siguientes posibles configuraciones finales para una asignación de tres registros se escogerá la opción de la izquierda:



Suma total => 16



Suma total => 15

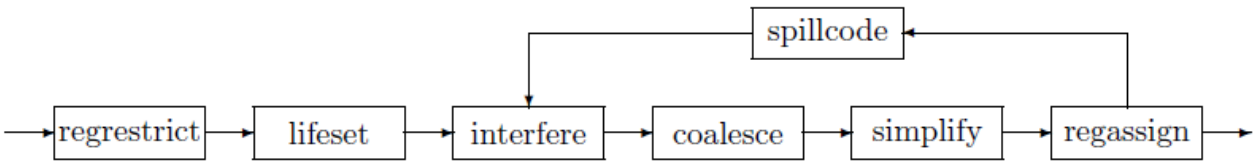
Finalmente, y una vez que los nuevos registros son conocidos, se sustituyen los registros originalmente asignados por los nuevos. El orden de asignación es importante, ya que para configuraciones que utilicen muchos registros es imprescindible alejar lo máximo posible variables con tiempos de vida muy largos que se encuentren en diferentes registros físicos. Por ello,

previamente se habrá calculado el tiempo de vida de cada una de las variables visitando el gráfico de control de flujo (CFG) del programa. Las variables se ordenarán en orden decreciente de tiempos de vida y se reasignarán atendiendo a dicho orden.

FLUJO DEL ALGORITMO

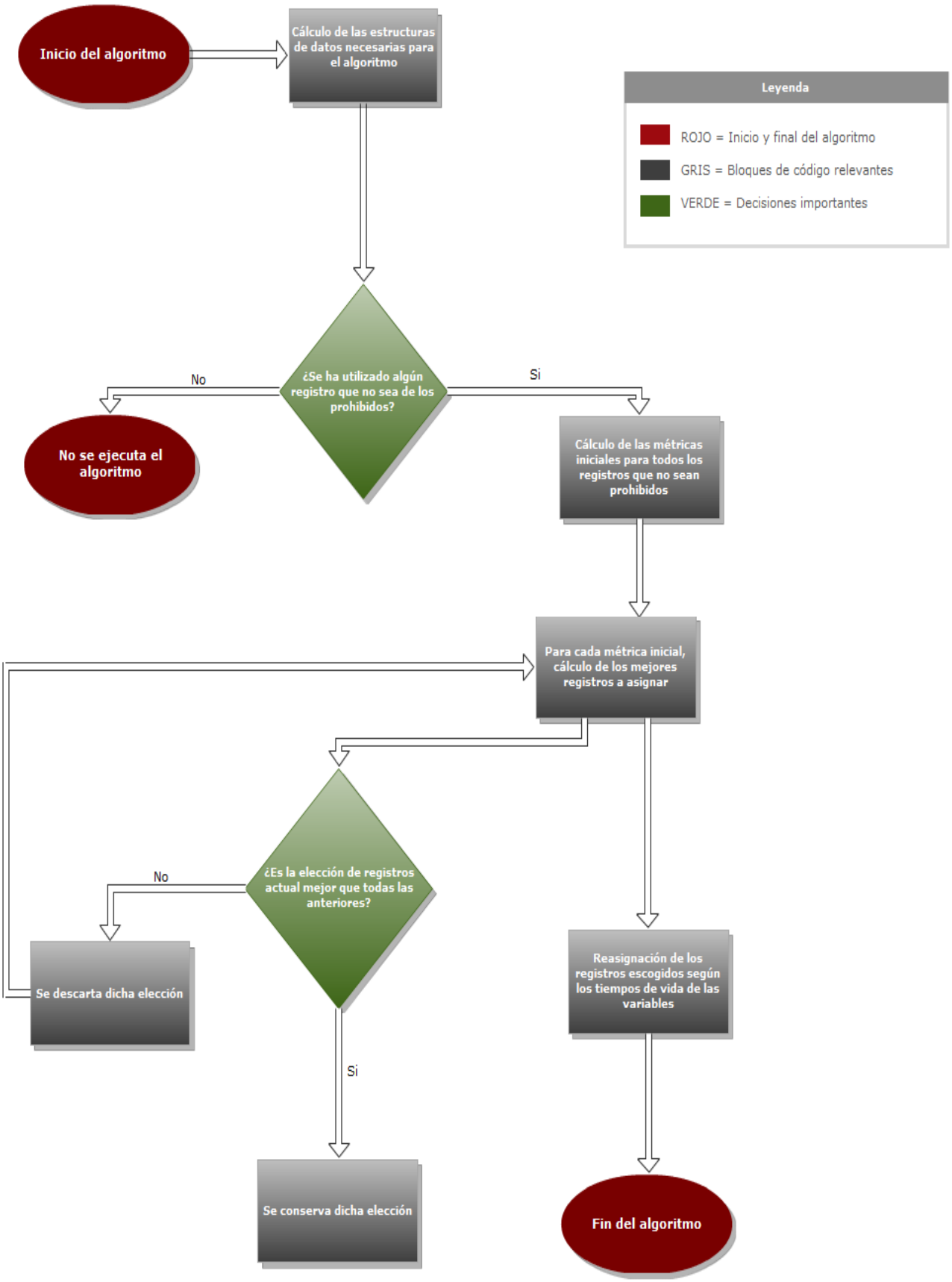
Vamos a desgranar los pasos que se llevan a cabo en el algoritmo, para que lo anteriormente descrito tome sentido. Para ello, mostraremos en primer lugar un grafo de flujo de ejecución del mismo, para posteriormente completarlo con un pseudocódigo simple.

El algoritmo propuesto ha sido implementado en el módulo de emisión de código del compilador (*emit*), en el que ya se posee la estructura que indica para cada variable en qué registro físico se va a almacenar. Dichas estructuras han sido previamente calculadas en el módulo de asignación de registros, que está formado entre otros por los siguientes motores:



Cabe destacar los módulos *lifeset*, cuya función es calcular los tiempos de vida de cada uno de los registros lógicos del programa, y *regassign,* cuya función es asignar registros físicos a cada uno de los registros lógicos del programa, rellenando una tabla con dicha información. Los datos que necesita *regassign* son previamente calculados por *interfere, coalesce* y *simplify*.

Tras finalizar el módulo de asignación de registros, comenzará la generación de código en el módulo *emit* de emisión de código. Previamente a dicha generación se llevará a cabo el algoritmo propuesto, con el fin de modificar la tabla que posee las asignaciones propuestas por el módulo *regassign*.


Inicio del algoritmo
Cálculo de las estructuras de datos necesarias para el algoritmo
Leyenda
ROJO = Inicio y final del algoritmo
GRIS = Bloques de código relevantes
VERDE = Decisiones importantes
¿Se ha utilizado algún registro que no sea de los prohibidos?
No
Si
No se ejecuta el algoritmo
Cálculo de las métricas iniciales para todos los registros que no sean prohibidos
Para cada métrica inicial, cálculo de los mejores registros a asignar
¿Es la elección de registros actual mejor que todas las anteriores?
No
Si
Se descarta dicha elección
Se conserva dicha elección
Reasignación de los registros escogidos según los tiempos de vida de las variables
Fin del algoritmo

Un pseudocódigo general del algoritmo podría ser el siguiente. Se supone que las estructuras necesarias han sido calculadas previamente a la ejecución del algoritmo:

```
// INICIO DEL ALGORITMO
Para cada registro del banco de registros{
    Si dicho registro no es prohibido{
        Inicializar su métrica, habiendo escogido como
            primer registro al actual;
        Actualizar la métrica actual, marcando en las
            métricas a los prohibidos;
    }
}
Para cada registro del banco de registros{
    Si dicho registro no es prohibido{
        Repetir{
            Recorrer la métrica en busca del mejor
        registro;
            Añadirlo a asignacionActual;
            Actualizar la métrica de todos los registros;
        }tantas veces como registros haya que asignar
        Calcular el valor de la suma total de métricas;
        Si dicho valor es mejor que los anteriores{
            asignacionFinal <- asignacionActual;
        }
    }
}
// EL SIGUIENTE BUCLE SE REALIZA SOBRE EL VECTOR
    DE VARIABLES ORDENADAS POR TIEMPOS DE VIDA
Para cada variable del programa{
    Obtener el registro en el que inicialmente estaba dicha
        variable;
    Asignar a dicha variable el primero de los registros de
        asignacionFinal;
    Modificar el resto de variables que inicialmente habían
        sido asignadas a dicho registro con el nuevo;
}
// FIN DEL ALGORITMO
```

VERIFICACIÓN DEL ALGORITMO

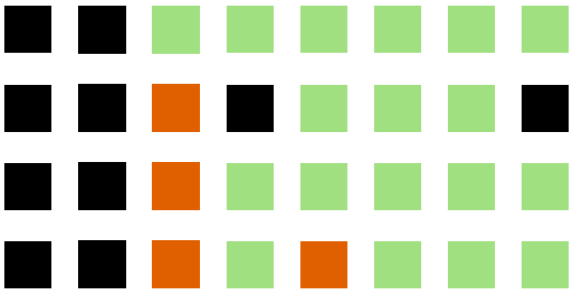
Vamos a mostrar los resultados que hemos obtenido al aplicar el algoritmo anterior sobre diversos bancos de pruebas, pequeños programas de código en C que reflejan de forma realista cómo se comportaría el algoritmo. Comprobaremos que se cumplen las expectativas y los objetivos propuestos, principalmente la optimalidad técnica de los resultados.

Cada uno de los bancos de pruebas posee sus peculiaridades, que serán comentadas en los siguientes apartados. Nuestro algoritmo nos ha permitido verificar su funcionalidad para diferentes estructuras de bancos de registros, tal y como se explicará a continuación.

BENCHMARK 1: ALGORITMO DE LAS TORRES DE HANOI

Como se puede comprobar en las siguientes figuras, el comportamiento inicial de compilar el código sin aplicar el algoritmo aglomeraba todos los registros escogidos por el compilador en una misma área determinada del banco de registros:

Nota: las siguientes figuras representan configuraciones de registros escogidos por el compilador, siendo los negros los prohibidos (no reasignables), los naranjas los asignados y los verdes los no utilizados.



Como se puede observar en la anterior figura, para un banco de registros de 32 registros como es el de la arquitectura SPARC y

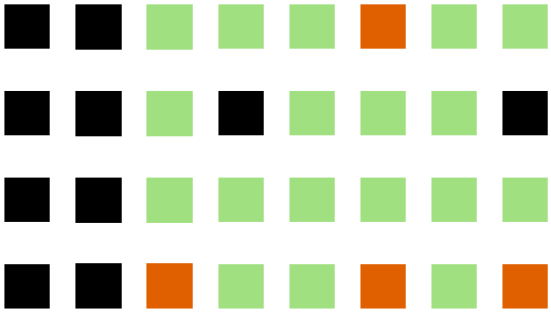
con una disposición de los registros en 4 filas y 8 columnas ocurre lo que se esperaba. Han sido utilizados cuatro de los registros del banco. Para las disposiciones de 2 filas con 16 columnas y de 1 fila con 32 columnas ocurre algo similar:

Aplicando nuestro algoritmo, los resultados son muy diferentes y, como era de esperar, óptimos. Verificaremos el funcionamiento de las métricas variando los pesos otorgados a distancia entre registros y a distancia de cada registro al borde del chip.

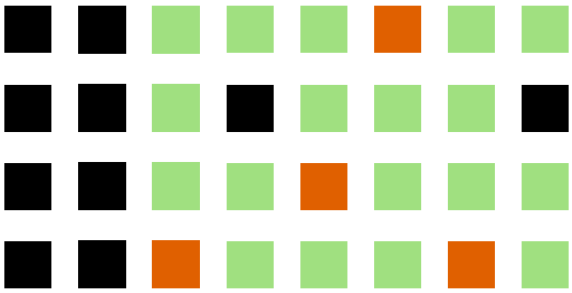
Nótese que las métricas pierden toda su relevancia en el caso de 2 y 1 fila de registros en el banco. Esto es debido a que la distancia de cada registro al borde del chip es en todos los casos cero, por lo que con cualquier peso que se indique el algoritmo devolverá los mismos resultados. Para dichas configuraciones los resultados han sido los siguientes:

Puede observarse que los registros escogidos se han espaciado enormemente, por lo que se ha ganado disipación de calor entre dichos registros. Veamos que ocurre en el caso de la arquitectura de 4 filas y 8 columnas, en la que las métricas cobran especial relevancia:

En el caso de otorgar algo de peso a la distancia de los registros al borde del chip, se puede observar en la siguiente figura que los registros se aislan del resto, asignándose a las zonas de la periferia del chip de silicio:



Sin embargo, tras otorgar un peso nulo a la distancia de los registros al borde del chip, el algoritmo encuentra que alguno de los mejores registros asignables se encuentra en la zona más interna del chip, como demuestra la siguiente figura:

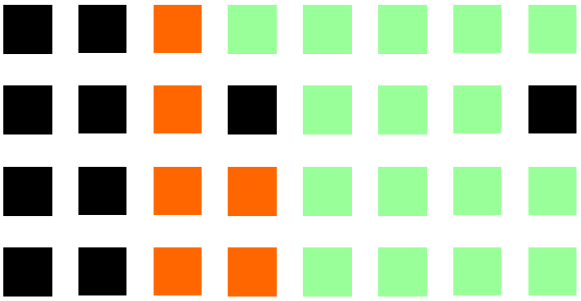


Dichos resultados verifican la optimalidad del algoritmo, ya que los registros se han distanciado y se ha eliminado el más que probable *"hotspot"* que se podría haber generado en el banco de registros sin la aplicación del algoritmo.

BENCHMARK 2: BUCLE COMPLEJO

En este nuevo caso de programa de prueba, el número de registros totales que se han necesitado ha sido 6, por lo que se comenzará a visualizar la potencia del algoritmo, algo que probablemente en un programa como el anterior que requería tan pocos registros era dificilmente visualizable.

El comportamiento inicial de compilar el código sin aplicar el algoritmo generaba, como se puede observar en las figuras siguientes, un mayor punto caliente en el banco de registros. Por ejemplo, en el caso de un banco de registros con 4 filas y 8 columnas se ve claramente este hecho:

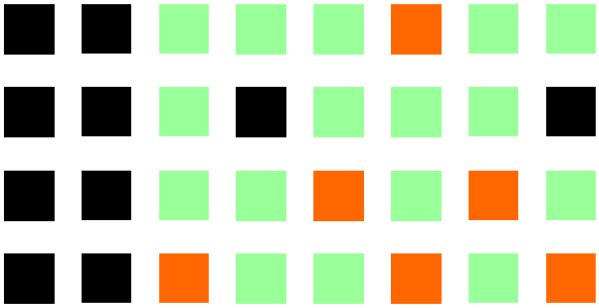


De igual forma ocurre con distintas estructuras del banco de registros:

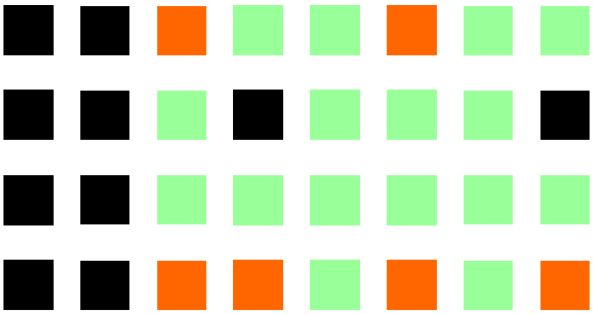
Nuevamente, tras aplicar nuestro algoritmo los resultados obtenidos han sido óptimos. Como se puede observar en las siguientes figuras, el algoritmo ha llevado a cabo una redistribución de los registros esparciéndolos a lo largo del chip de silicio, por lo que nuevamente se conseguirá una gran disipación del calor en el chip y se eliminará el hotspot que se había generado. Veámoslo en las figuras:

En este caso, se puede observar que el algoritmo ha dejado entre los registros escogidos el suficiente espacio como para que siempre exista una vía de escape para el calor. Lo mismo ocurrirá en el siguiente banco de registros:

En el caso del banco de 4x8 registros, vuelven a ser relevantes los pesos otorgados a cada una de las distancias. En el caso de otorgar poco o nulo peso a la distancia al borde del chip, los registros escogidos por el algoritmo están situados uniformemente en el banco de registros:

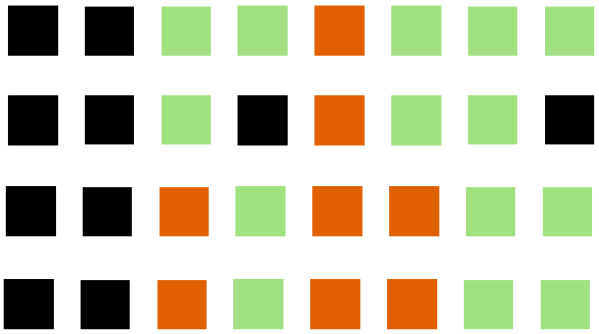


Sin embargo, en caso de otorgar mayor peso a la distancia al borde el algoritmo sitúa de nuevo a los registros en la periferia del chip de silicio, por lo que a costa de conseguir que en el centro del chip no se asignen registros la periferia puede llenarse de registros. Veámoslo en la figura:



BENCHMARK 3: NÚMEROS PRIMOS

El siguiente de los bancos de prueba utilizados ha sido el algoritmo que encuentra los N primeros números primos **[20]**. En este caso el número de registros físicos utilizados ha sido de 8 por lo que, como se podrá comprobar en las siguientes figuras, se han generado 2 hotspots claramente diferenciados. Veamos dichos puntos en el banco de 4 filas por 8 columnas:



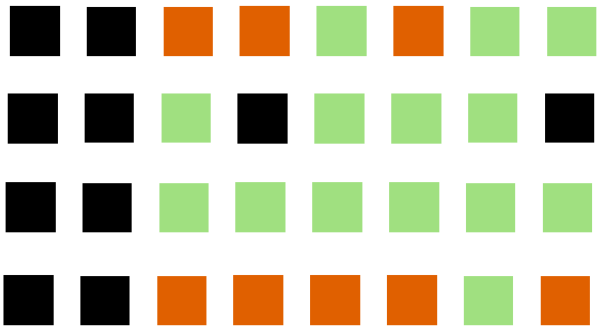
Dichos puntos se hacen aún más evidentes en el caso de los bancos de registros con 2 y 1 fila correspondientemente. Las imágenes siguientes lo demuestran:

Vamos a observar cómo influye la aplicación de nuestro algoritmo en este caso. Para los casos en los que los pesos no influyen en la métrica, se puede observar una evidente dispersión de los registros en el banco. En el caso, por ejemplo, del banco de registros de una única fila de registros se puede observar cómo se han salteado a lo largo del chip.

Otro caso de especial interés es el caso del banco de registros con dos filas de registros. En este caso la dispersión

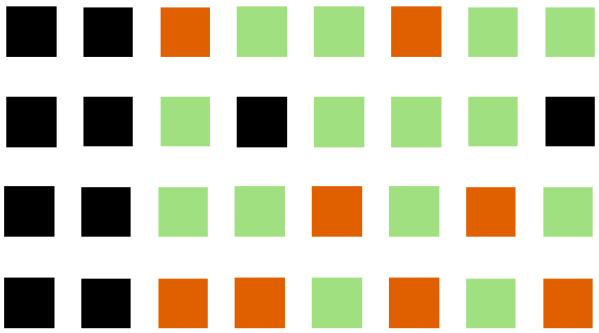
también es más que evidente. Sin embargo, en cierto momento el algoritmo ha de escoger registros cuya métrica ya no es elevada. Por ello ocurre el posible hotspot que se observa en la parte izquierda de la figura. Sin embargo, gracias a cómo ha sido diseñado el algoritmo dicho hotspot se minimiza, ya que el orden de asignación de los registros en el banco viene dado por los tiempos de vida de las variables. Por ello, los registros de la zona izquierda se asignarán a las variables con menor tiempo de vida, por lo que el efecto del calor en dicha zona se minimizará.

Veamos ahora el efecto de los pesos en el banco de registros de cuatro filas y ocho columnas. En este caso, al tratarse de un número considerable de registros a asignar hemos creído conveniente el mostrar más casos de diferenciación de pesos. En primer lugar, mostraremos la configuración escogida por el algoritmo cuando se otorga el mismo peso a la bonificación por distancias y a la penalización por distancia al borde del chip:

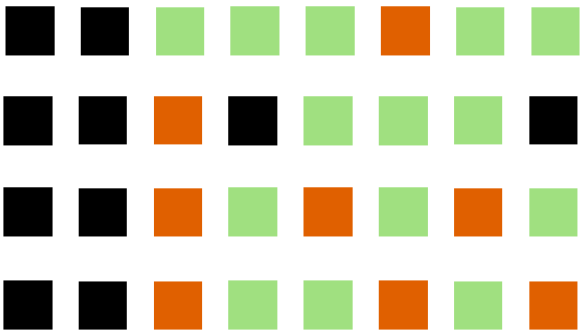


Como se muestra en la imagen anterior, se observa una evidente masificación de los registros en el borde del chip. Esto ocurre debido a la alta importancia que se ha otorgado a la distancia al borde del chip. La consecuencia evidente de ello ha sido que la

zona central del banco de registros ha quedado vacía de registros. Por ello, vamos a ajustar el peso de dicha distancia al centro del chip reduciéndolo ligeramente. La configuración obtenida de realizar dicho cambio ha sido la siguiente:



En este caso, la transformación ha sido sumamente positiva, ya que existe una mayor separación de los registros en el banco que con la anterior elección de pesos, a costa de que varios de ellos se encuentren en la zona central del chip. Para finalizar con este benchmark, vamos a observar cómo afecta a la elección de los registros el no otorgarle a la distancia al borde del chip ningún peso. El resultado se muestra en la siguiente imagen:



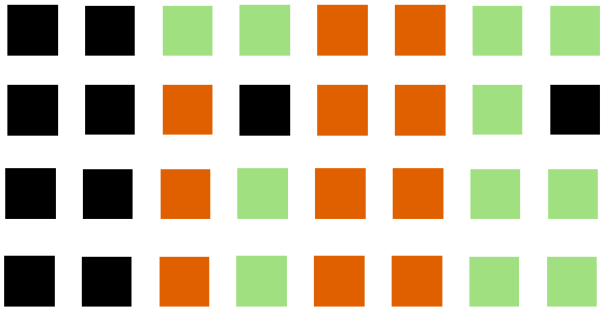
La diferencia principal con el caso anterior viene dada por el hecho de que no exista penalización en los registros centrales del banco de registros. Por esa razón, en el momento en que registros centrales y periféricos han obtenido valores de métrica idénticos el algoritmo los ha escogido en orden de aparición. Por ello es importante a la hora de ejecutar el algoritmo ajustar correctamente

los valores de peso, y cuanto mayor número de registros estén en juego más patente se hace dicha necesidad.

BENCHMARK 4: CÁLCULO DE Π POR PROBABILIDAD

El último de los bancos de prueba utilizados ha sido el algoritmo que computa el valor de Pi por probabilidad **[21]**. Este algoritmo hace uso de once registros físicos, la mitad de los registros disponibles en el banco para reasignar. Por ello, como se podrá observar en las siguientes imágenes, los *hotspots* han aumentado todavía más de tamaño.

En el caso del banco de registros de 4x8 registros, el calor que generará la configuración por defecto en el chip puede llegar a ser peligrosa, ya que varios registros centrales del banco consecutivos están siendo utilizados, y en dichos registros el calor no tiene una vía de escape, debido a que los registros que los rodean también están siendo utilizados por el programa.



El primero de los casos que vamos a estudiar es el del banco de registros con una única fila de registros. Como se puede observar en la siguiente figura, el algoritmo ha salteado los registros a lo largo del banco de registros.

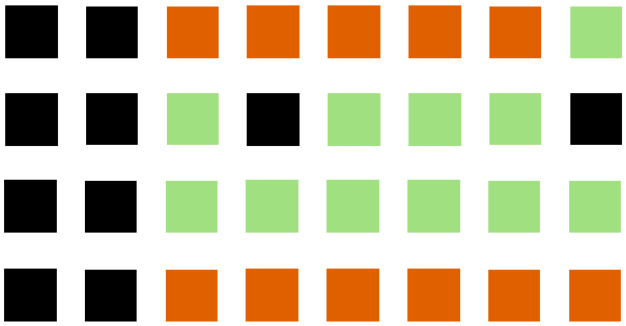
En el momento en que no han quedado registros para intercalar, el comportamiento del algoritmo ha consistido en escoger los primeros disponibles en el banco (los de la parte izquierda del mismo), ya que todos los restantes poseían la misma métrica. Esto no es problema, ya que, como se ha indicado anteriormente, sobre dichos registros van a recaer las variables con menor tiempo de vida.

A continuación, veremos que en el caso del banco de registros con dos filas el comportamiento del algoritmo es similar:

De nuevo, el algoritmo no ha tenido más remedio que escoger registros muy próximos en un área del banco de registros, aunque

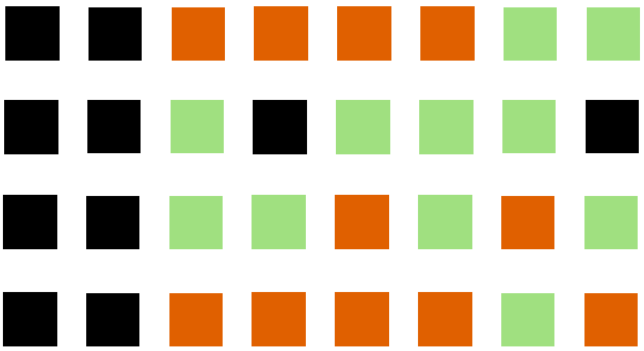
dicho problema se minimiza gracias a los tiempos de vida de las variables.

Vamos a estudiar de nuevo el efecto de los pesos sobre los valores de distancia. En primer lugar, vamos a observar cómo en este benchmark ocurre lo mismo que en los anteriores cuando se penaliza mucho la distancia al borde del chip (es decir, se otorga un gran peso a dicho valor):

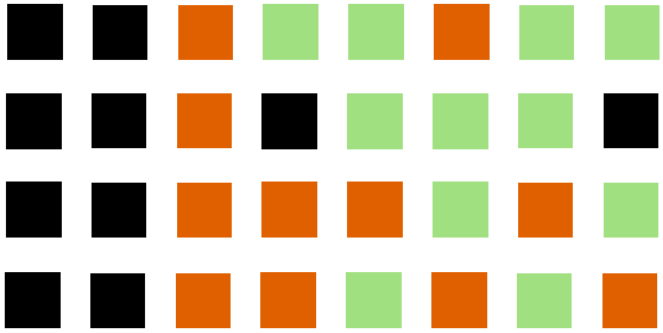


De nuevo, la zona central del chip se ha vaciado de registros escogidos, mientras que los bordes del mismo se han sobrecargado. En todo momento ha de tenerse en cuenta que el orden en que los registros sean asignados dependerá de los tiempos de vida de las variables, por lo que el efecto de los hotspots se minimizará lo máximo posible.

Seguidamente, observamos cómo afecta el reducir la penalización por distancia al borde del chip, obteniendo una configuración más uniforme:



Como se observa en la figura, se ha ganado espacio en los bordes del chip, reduciéndose el calor en dicha zona. El algoritmo escoge en este caso algunos registros centrales. Sin embargo, la pequeña penalización que se impone a dicha zona del chip implica que siga recargándose el área periférica del chip. Por ello, eliminamos dicha penalización, lo que desemboca en el siguiente conjunto de registros:



De nuevo, el algoritmo ha vuelto a diversificar la colocación de los registros a lo largo del chip, permitiendo que los bordes del mismo queden menos ocupados. La consecuencia de ello ha sido que cierta zona del banco de registros ha quedado completamente ocupada de registros, efecto que no nos conviene obtener y que, por lo tanto, evitaremos.

CONCLUSIÓN DEL ESTUDIO DEL ALGORITMO

Generalmente, el algoritmo ha devuelto resultados similares para todos los bancos de pruebas sobre los que se ha ejecutado. Hemos observado que los pesos para las métricas influyen enormemente en la elección de registros que se lleva a cabo en el algoritmo. Asi, hemos observado que para penalizaciones muy grandes o nulas de la distancia de cada registro al borde del chip, el algoritmo otorga en ciertos casos resultados no deseables, como es la masificación de ciertas zonas del banco de registros.

Una elección adecuada de dichos pesos nos permitirá obtener unos mejores resultados, por lo que se debe de penalizar lo justo a la distancia al borde del chip y beneficiar igualmente lo justo a la distancia entre pares de registros.

Una carencia evidente de este algoritmo es el hecho de que es imposible controlar si varias variables con tiempos de vida muy elevados han sido asignadas al mismo registro. Esto ocurre porque nuestra reasignación se basa en la asignación inicial que se ha llevado a cabo en la fase de *register assignment.* Por ello, nos vimos en la necesidad de mejorar el algoritmo para suplir dicha carencia, teniendo más en cuenta tiempos de vida de las variables del programa.

5. ASIGNACIÓN BASADA EN TIEMPOS DE VIDA

OBJETIVO DEL ALGORITMO

El segundo de los algoritmos propuestos viene motivado por la carencia evidente que el primero de los algoritmos sufría. Se vio necesario el replanteamiento del algoritmo inicial, con el fin de que variables con tiempos de vida muy elevados y que coincidieran en el tiempo se encontrasen lo más alejadas posibles.

El nombre de este algoritmo proviene del hecho de que la manera en que el algoritmo escoge los registros no es determinista. Es decir, cada programa tendrá su propia asignación, independientemente del número de registros utilizados por la asignación inicial y totalmente dependiente de los **tiempos de vida** de cada variable del programa y de los **solapamientos** entre las mismas.

En el momento de diseñar un algoritmo tan diferente al anterior, nos planteamos la forma de poder reutilizar lo máximo posible el código del algoritmo anterior. Por ello, una serie de conceptos se conservan respecto del anterior algoritmo.

En primer lugar, este algoritmo consiste igualmente en un problema de **optimización multiobjetivo**. Los objetivos que han debido de cumplirse en todo momento se han conservado respecto del algoritmo original:

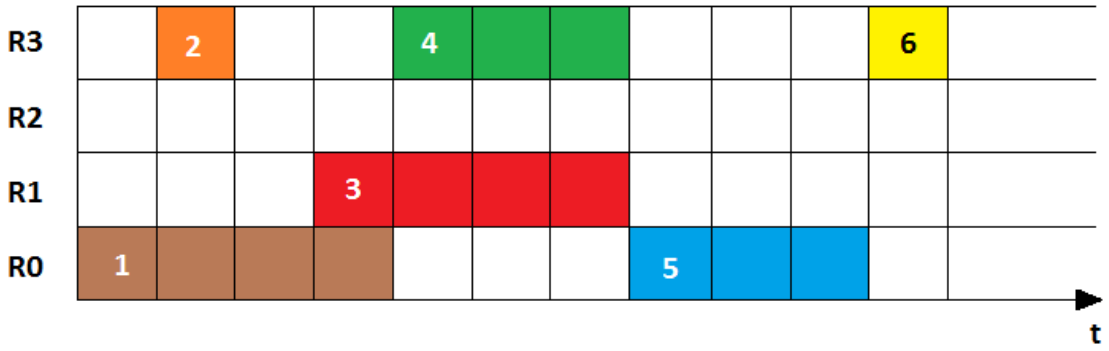
- La distancia entre cada par de registros escogido ha de ser la máxima posible.
- La distancia de cada registro escogido al borde del chip de silicio ha de ser la mínima posible.

Igualmente se han conservado los conceptos de métrica basada en los objetivos anteriores, de pesos para cada uno de

dichos valores y la forma en la que se calculan los mejores registros a asignar en todo momento. Así, el núcleo del algoritmo es idéntico al algoritmo de asignación estática, con ligeras diferencias que explicaremos a continuación.

Como se ha comentado anteriormente, las variables que posee cada programa serán el eje sobre el que girará el algoritmo. Por ello, será necesario conocer para cada registro lógico su grado y los registros lógicos con los que solapa en el tiempo.

El **grado de solapamiento** de un pseudoregistro se define como el porcentaje del tiempo de vida que solapa con cualquiera de los restantes pseudoregistros. Es decir, un registro lógico que durante su tiempo de vida no coincida con ninguno de los restantes registros lógicos tendrá un grado de solapamiento cero, mientras que un registro lógico cuyo tiempo de vida esté totalmente solapado tendrá grado uno. Veamos un ejemplo gráfico:



- El pseudoregistro 1, con tiempo de vida 4, posee un grado de 0,5 y solapa temporalmente con 2 y 3.
- El pseudoregistro 2, con tiempo de vida 1, posee un grado de 1 y solapa temporalmente con 1.
- El pseudoregistro 3, con tiempo de vida 4, posee un grado de 1 y solapa temporalmente con 1 y 4.
- El pseudoregistro 4, con tiempo de vida 3, posee un grado de 1 y solapa temporalmente con 3.
- El pseudoregistro 5, con tiempo de vida 3, posee un grado de 0 y no solapa temporalmente con nadie.

- El pseudoregistro 6, con tiempo de vida 1, posee un grado de 0 y no solapa temporalmente con nadie.

El grado es importante en el algoritmo propuesto, ya que definirá el orden en que se asignarán registros físicos a las variables del programa. Los registros lógicos con **mayor grado** (es decir, los que sufren mayor solapamiento) serán los más preferentes, mientras que los de menor grado (los menos solapados) se asignarán al final ya que son los que menor conflictos causan.

Sin embargo puede haber casos, como en el ejemplo anterior, en el que varios registros lógicos posean un grado elevado. La forma que utiliza el algoritmo para escoger cuál de los pseudoregistros implicados ha de ser tratado en primer lugar es multiplicando su valor de grado por el número total de pseudoregistros con los que solapa. Puede ocurrir que un registro lógico con un grado muy elevado solape con muchas variables temporalmente. El algoritmo dará prioridad a casos con un gran número de solapamientos temporales frente a casos con un grado elevado y con poco solapamiento. En el ejemplo anterior, 3 tendrá preferencia sobre 2 y 4, al solapar aquél con otros 2 registros lógicos y éstos únicamente con uno.

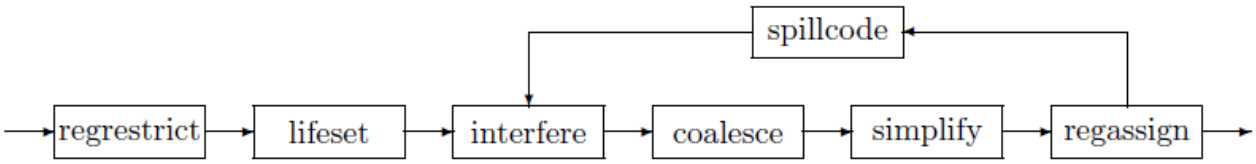
Cuando se tienen todos esos datos computados, basta con que el algoritmo lleve a cabo la reasignación estática de dichos registros lógicos, tomando como número de registros totales a colocar a lo largo del chip de silicio el número de registros lógicos que se han visto implicados en el solapamiento.

Existe un problema en este planteamiento: cuando se diseñó el algoritmo estático, se partió de la premisa de que el algoritmo nunca se encontraría con casos en los que tuviese que asignar más registros que registros disponibles en el banco de registros. Sin embargo, es imposible conocer cuántos registros lógicos solaparán temporalmente con un registro determinado. Por ello, fue imprescindible rediseñar dicha parte del algoritmo para contemplar esa situación, llevando a cabo la asignación en varios pasos.

FLUJO DEL ALGORITMO

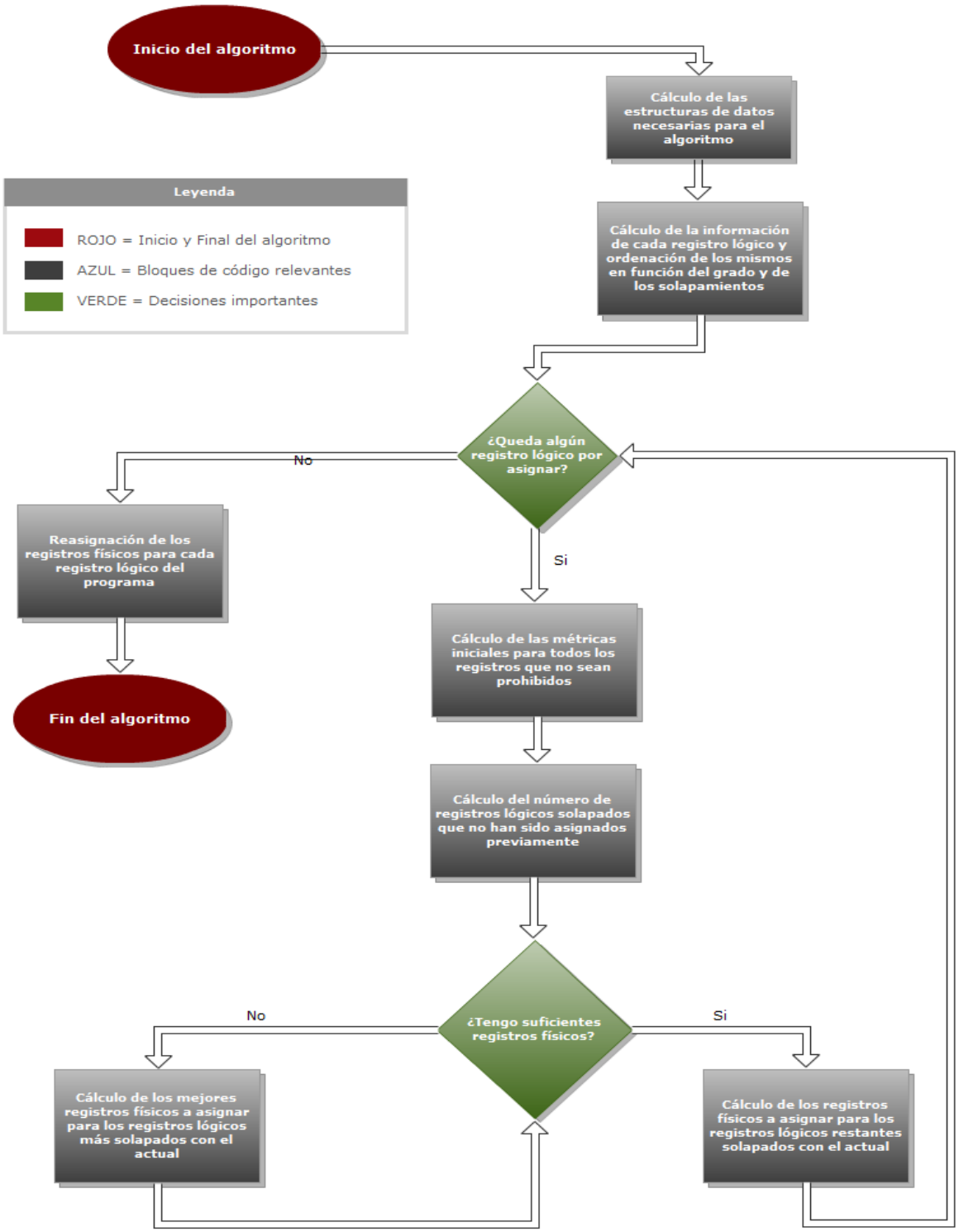
Vamos a desgranar los pasos que se llevan a cabo en el algoritmo, para que lo anteriormente descrito tome sentido. Para ello, mostraremos en primer lugar un grafo de flujo de ejecución del mismo, para posteriormente completarlo con un pseudocódigo simple.

El algoritmo propuesto ha sido implementado nuevamente en el módulo de emisión de código del compilador (*emit*), en el que ya se posee la estructura que indica para cada variable en qué registro físico se va a almacenar. Dichas estructuras han sido previamente calculadas en el módulo de asignación de registros, que está formado entre otros por los siguientes motores:



Cabe destacar los módulos *lifeset*, cuya función es calcular los tiempos de vida de cada uno de los registros lógicos del programa, y *regassign,* cuya función es asignar registros físicos a cada uno de los registros lógicos del programa, rellenando una tabla con dicha información. Los datos que necesita *regassign* son previamente calculados por *interfere, coalesce* y *simplify*.

Tras finalizar el módulo de asignación de registros, comenzará la generación de código en el módulo *emit* de emisión de código. Previamente a dicha generación se llevará a cabo el algoritmo propuesto, con el fin de modificar la tabla que posee las asignaciones propuestas por el módulo *regassign*.


Inicio del algoritmo
Cálculo de las estructuras de datos necesarias para el algoritmo
Leyenda
ROJO = Inicio y Final del algoritmo
AZUL = Bloques de código relevantes
VERDE = Decisiones importantes
Cálculo de la información de cada registro lógico y ordenación de los mismos en función del grado y de los solapamientos
¿Queda algún registro lógico por asignar?
No
Si
Reasignación de los registros físicos para cada registro lógico del programa
Fin del algoritmo
Cálculo de las métricas iniciales para todos los registros que no sean prohibidos
Cálculo del número de registros lógicos solapados que no han sido asignados previamente
¿Tengo suficientes registros físicos?
No
Si
Cálculo de los mejores registros físicos a asignar para los registros lógicos más solapados con el actual
Cálculo de los registros físicos a asignar para los registros lógicos restantes solapados con el actual

Un pseudocódigo general del algoritmo podría ser el siguiente. Se supone que las estructuras necesarias han sido calculadas previamente a la ejecución del algoritmo:

```
Para cada registro lógico del programa {
    Calcular su grado;
    Calcular los registros lógicos que solapan con él en el
        tiempo;
}
Ordenar los registros lógicos en grado y solapamiento;
Mientras queden registros lógicos a asignar {
    Ordenación de los registros lógicos solapados con el
        actual por tiempos de solapamiento;
    Inicialización de las métricas;
    Calcular el número total de registros lógicos a asignar
        teniendo en cuenta los previamente asignados;
    Actualización de las métricas con los registros lógicos ya
        asignados;
    Si el número de registros lógicos a asignar es menor o
        igual al número de registros disponibles del banco
        de registros{
        Aplicar la asignación del algoritmo estático;
        Actualizar el registro físico asignado a cada
            registro lógico;
    }
    si no{
        Mientras queden registros solapados con el actual{
            Escoger los registros lógicos más solapados
                con el actual;
            Aplicar la asignación del algoritmo estático;
            Actualizar el registro físico asignado a cada
                registro lógico;
            Actualizar las métricas con los registros
                escogidos;
        }
    }
}
```

VERIFICACIÓN DEL ALGORITMO

Tal y como se hizo con el algoritmo previo, demostraremos la optimalidad técnica del algoritmo estudiando diversos resultados sobre bancos de prueba disponibles. En este caso, el estudio realizado será ligeramente diferente al anterior ya que resulta irrelevante e innecesario mostrar los registros que ha escogido el algoritmo a lo largo del programa.

Esto es así porque, realmente, la elección de unos registros u otros dependerá exclusivamente del instante temporal del programa, y siendo un resultado indeterminista en muchos casos todos los registros del banco serán asignados a alguna variable en algún momento. Por ello, en este caso los resultados mostrados serán **termogramas**, imágenes que revelarán instantes temporales determinados del grafo de control de flujo del programa, mostrando qué registros están siendo utilizados en dicho momento y la temperatura actual de dicho registro en dicho instante de tiempo.

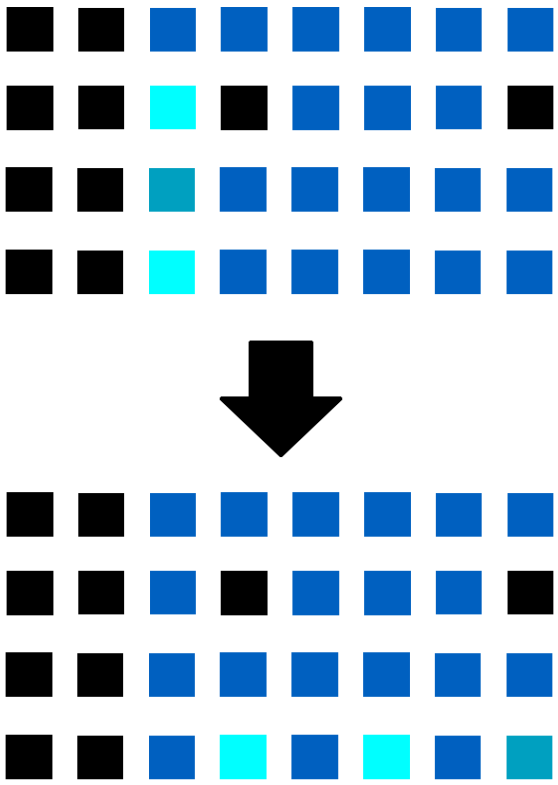
Gráficamente, las temperaturas se mostrarán en una escala de color entre el azul, correspondiente a los registros fríos, y el rojo, que corresponderá a los registros con una temperatura elevada. Gracias a esta representación, podremos visualizar cómo variables con tiempos de vida elevados y solapados han sido separadas en el banco de registros.

Por otra parte, mostraremos los resultados únicamente para el banco de registros de 4 filas por 8 columnas de registros, con unos pesos para la distancia entre pares ligeramente superior al peso para la distancia al borde del chip, ya que con el anterior banco de pruebas se demostró que devolvía la asignación más eficiente.

BENCHMARK 1: ALGORITMO DE DIJKSTRA

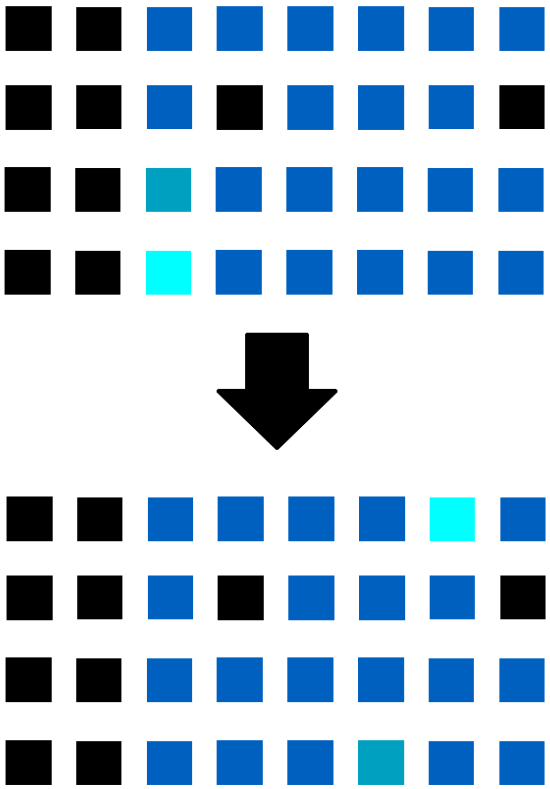
El primero de los bancos de prueba utilizados ha sido el algoritmo de caminos mínimos de Dijkstra **[22]**.

Hemos tomado valores en ciertos instantes temporales de ejecución del programa previamente a la aplicación del algoritmo, y los hemos comparado con los valores que la aplicación de nuestro algoritmo ha devuelto en los mismos instantes temporales. Las siguientes figuras muestran dicha diferenciación:

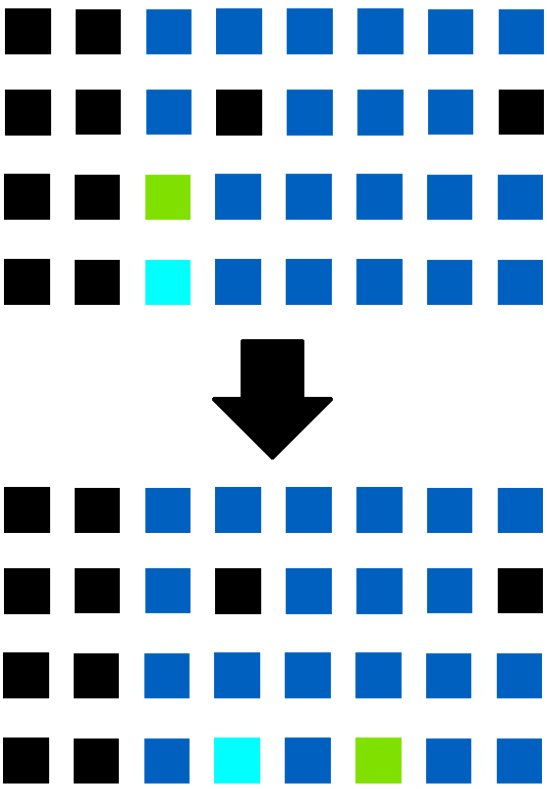


Como se puede observar en la figura, inicialmente la fase de *register assignment* había escogido registros muy cercanos y limitados en un área concreta del chip. Tras aplicar el algoritmo propuesto, los registros se han dispersado a lo largo del banco de registros, además de haberse situado en el borde del chip, lo que otorgará una mejor disipación del calor al estar en contacto con menor material.

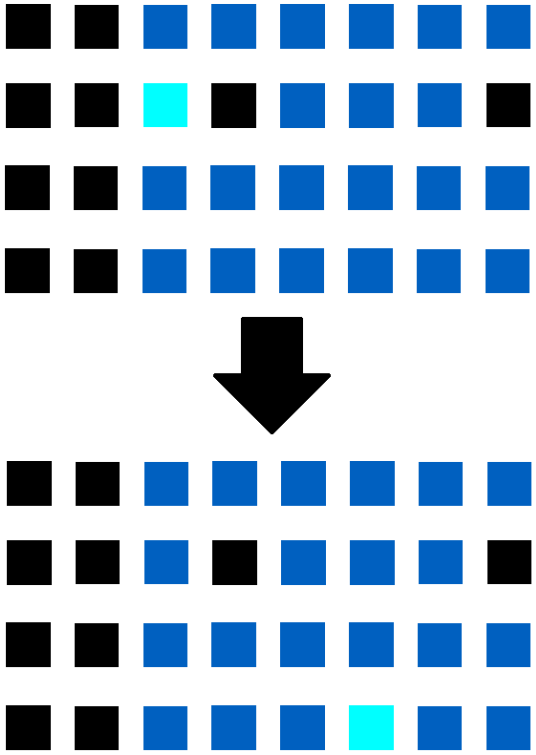
Veamos otros instantes temporales, y cómo el algoritmo ha resuelto la asignación de los registros utilizados:



De nuevo, se observa la separación que el algoritmo ha conseguido entre registros que inicialmente se encontraban igualmente situados. Veamos un ejemplo con un registro más caliente:



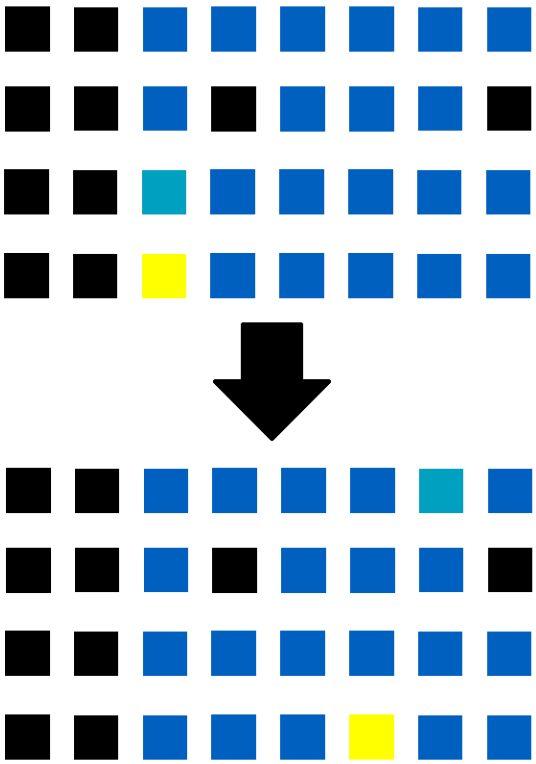
El siguiente ejemplo muestra un registro cuya variable asociada en dicho instante temporal no solapa con ninguna otra. Veamos cómo el algoritmo ha aislado dicha variable en el banco de registros:



BENCHMARK 2: ALGORITMO DE LAS TORRES DE HANOI

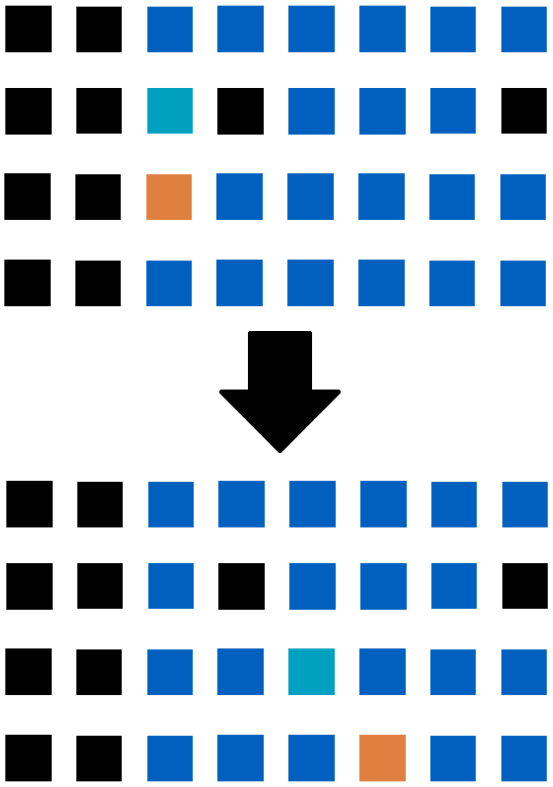
En este nuevo ejemplo de banco de prueba, vamos a observar casos de registros lógicos con tiempos de vida más elevados que en el banco de prueba anterior, lo que implicará registros con una temperatura más elevada.

El primero de los ejemplos expuestos muestra un instante temporal en el que uno de los registros comienza a alcanzar una temperatura considerable, debido a que el número de ciclos de reloj que han transcurrido desde el momento que dicho registro comenzó a utilizarse es elevado.

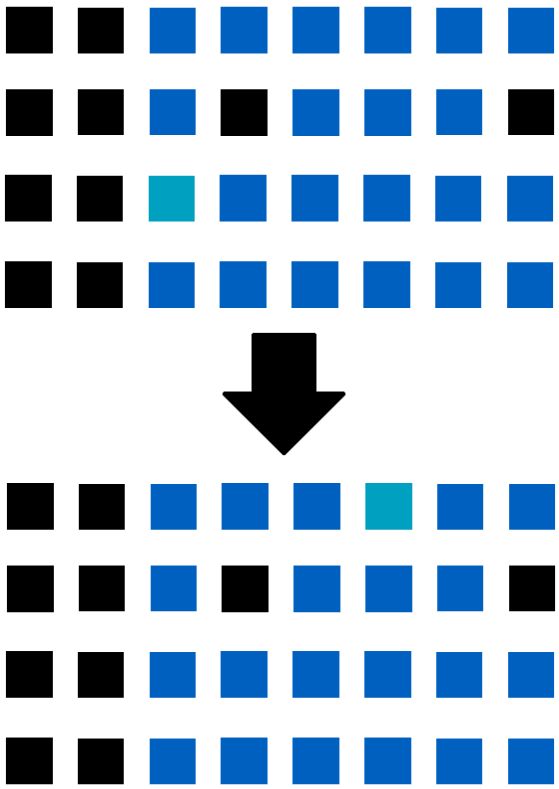


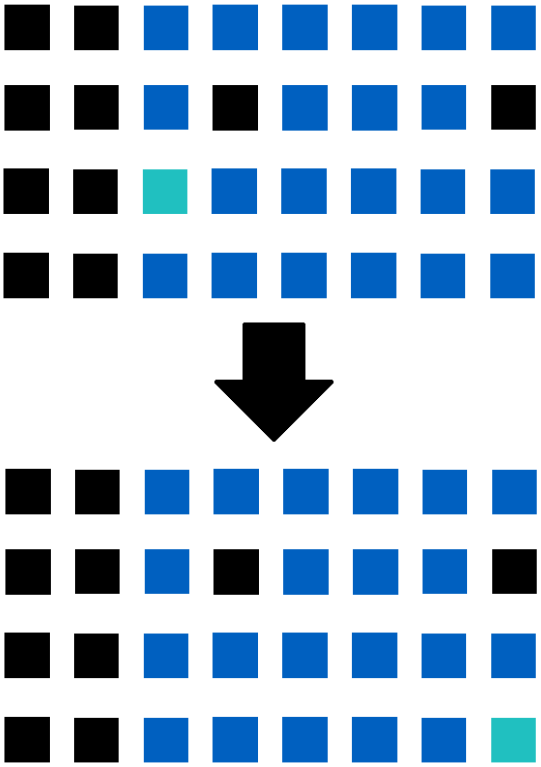
Se puede visualizar claramente cómo el algoritmo ha colocado estratégicamente al registro con mayor temperatura en la posición más óptima, mientras que el más frío se ha alejado a una mejor posición. Dicha reasignación homogeneíza claramente la temperatura en el chip, eliminando el hotspot que se hubiera generado en el chip.

Veamos otro ejemplo con un registro a una mayor temperatura. En este caso, el algoritmo no ha podido alejar los registros en dicho instante de tiempo lo máximo posible, ya que la variable que está almacenada en el registro de color anaranjado tenía un valor elevado de solapamiento. Sin embargo, que dichas variables hayan sido asignadas en registros tan cercanos implica que el tiempo de solapamiento común a ambas es muy reducido, por lo que el efecto del calor debido a dicha cercanía va a ser mínimo.

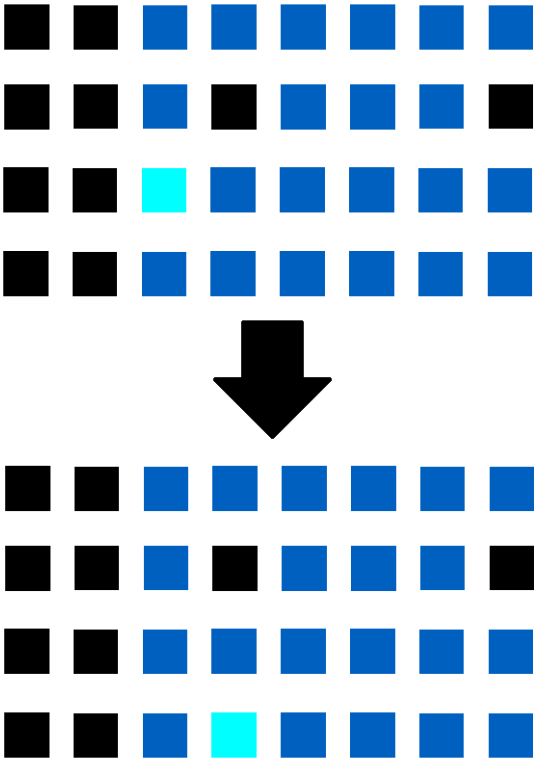


Veamos una serie de ejemplos de asignación eficiente de una única variable, importantes para demostrar el no determinismo del algoritmo en distintos instantes temporales:





El último de los ejemplos mostrará como para una variable que inicialmente se encontraba en el mismo registro que en los casos anteriores se ha realizado una asignación totalmente diferente:

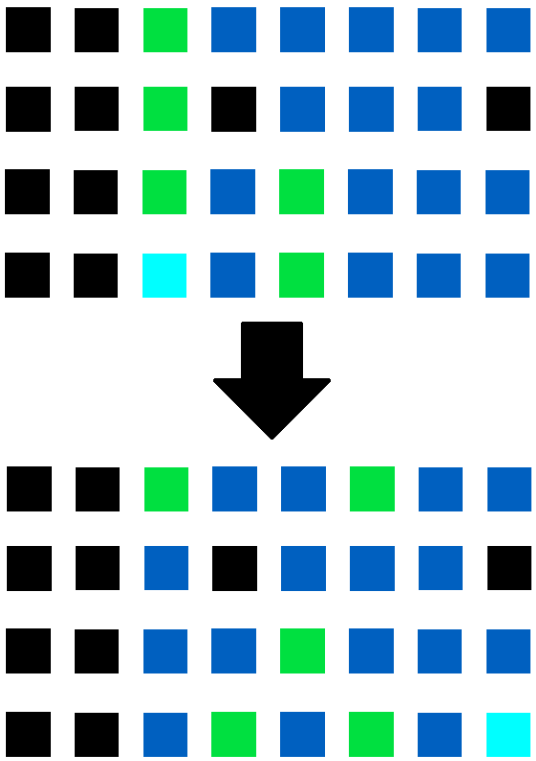


BENCHMARK 3: CÁLCULO DE Π POR PROBABILIDAD

El siguiente de los bancos de prueba utilizados es el algoritmo que computa el valor de Pi por probabilidad, anteriormente utilizado como banco de prueba en el primer algoritmo propuesto. Este caso es más interesante que los anteriores, ya que ciertas variables poseen un tiempo de vida extremadamente alto, que conseguirán temperaturas elevadas en el banco de registros.

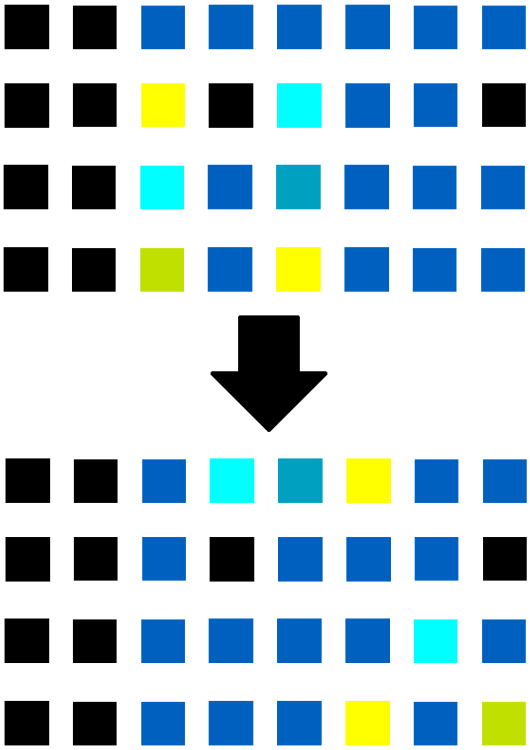
Por otra parte, este algoritmo posee ciertos registros lógicos con un número de solapamientos muy elevado, por lo que el algoritmo tiene que dividir la asignación como anteriormente se explicó. Los números tan elevados de solapamientos son consecuencia directa de los largos tiempos de vida de ciertas variables.

Veamos el primero de los ejemplos de distruibución de los registros que ha llevado a cabo el algoritmo en cierto instante temporal:

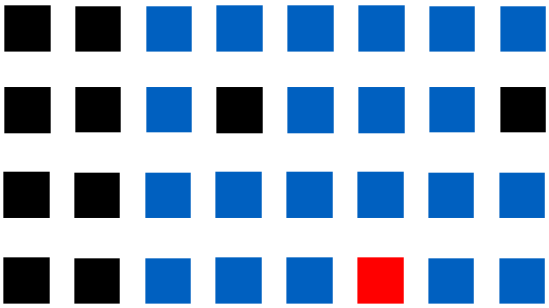


En este caso, comienza a demostrarse la potencia del algoritmo a la hora de eliminar puntos calientes en el chip, ya que se han separado los registros a mayor temperatura a lo largo del banco de registros, quedando zonas disponibles para que el calor se disipe correctamente.

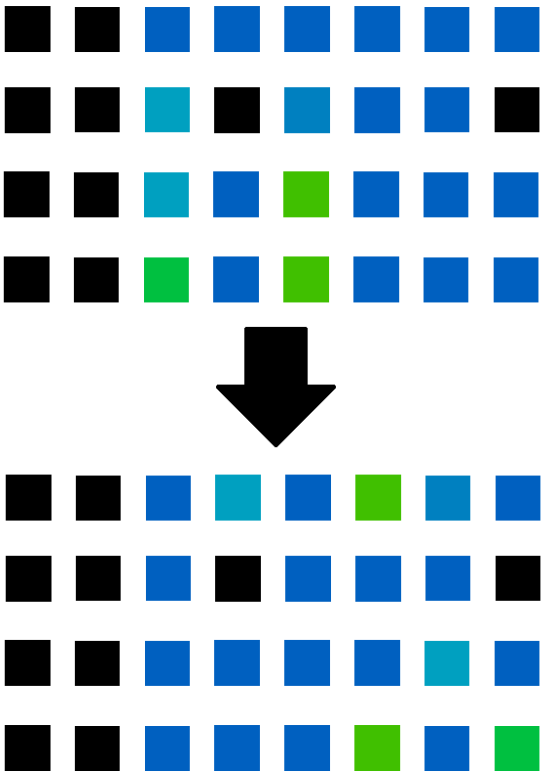
El siguiente ejemplo muestra la evolución temporal directa de la figura anterior, pudiéndose observar cómo ciertos registros han ido adquiriendo más temperatura mientras nuevas asignaciones de registros se han llevado a cabo.



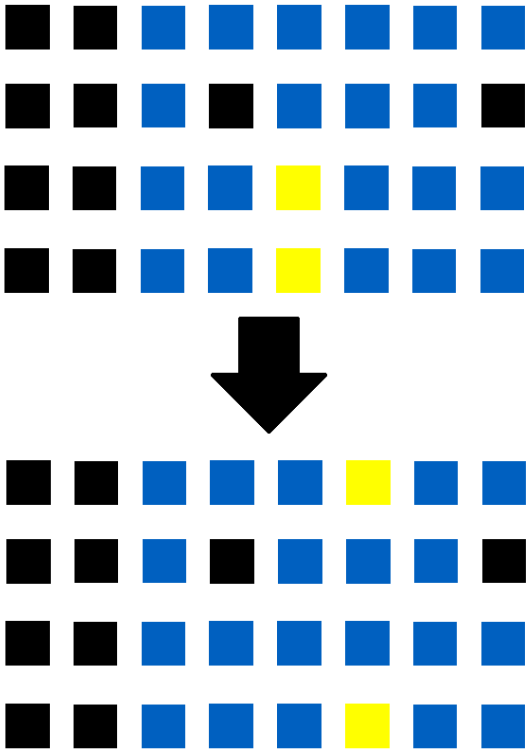
Las nuevas variables han sido asignadas en nuevos registros físicos, conservándose la optimalidad de la solución. Si seguimos temporalmente a la variable con mayor tiempo de vida de todas ellas, encontraremos que en cierto momento del programa el registro que la contiene poseerá una temperatura muy elevada. De no haber aplicado el algoritmo, se hubiese generado un *hotspot* en la zona más a la izquierda del banco de registros. Sin embargo, el aplicando el algoritmo el efecto de dicho hotspot se ha minimizado al máximo, como demuestra la siguiente figura:



Veamos otro ejemplo en el que ocurre algo similar. En este caso, podemos observar de nuevo cómo el algoritmo ha escogido registros diferentes, beneficiando así la disipación de calor en el banco de registros.



Temporalmente, podemos de nuevo observar cómo los registros en los que se encuentran almacenadas las variables con mayores tiempos de vida continuan adquiriendo temperatura. Sin embargo, el algoritmo ha escogido los lugares del banco de registros más adecuados para que se produzca una mayor disipación del calor:



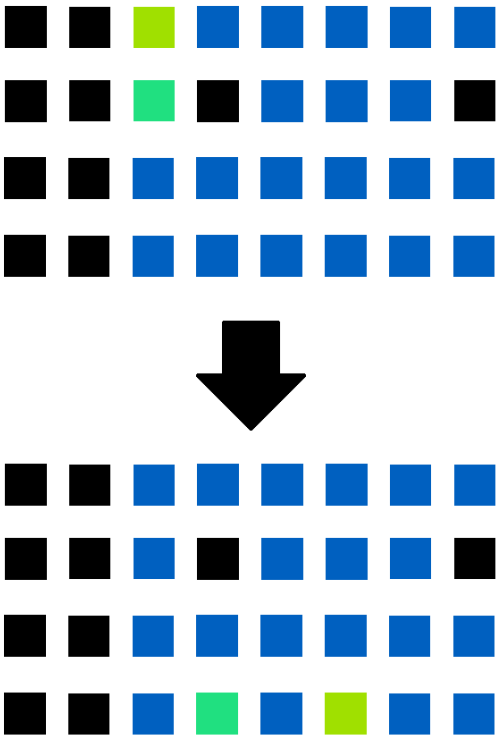
BENCHMARK 4: CRIBA DE ERATÓSTENES

El siguiente de los bancos de prueba utilizados ha sido el famoso algoritmo de la Criba de Eratóstenes, que encuentra los números primos menores que N.

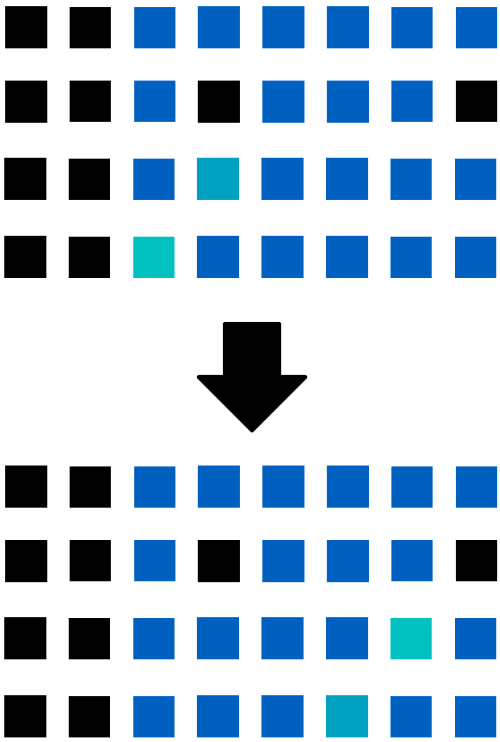
Este ejemplo tiene cierto interés, ya que en él se puede visualizar claramente cómo una variable del programa con un tiempo de vida muy elevado comienza a calentar cierto registro del banco, mientras que el resto de variables con tiempos de vida muy reducidos son asignadas a registros que se encuentran alrededor, de tal forma que las que poseen tiempos de vida más elevados se alejan más en el banco y las más reducidas se acercan más a dicho registro.

Sin embargo, es conveniente observar previamente un nuevo ejemplo de optimalidad del algoritmo, dadas dos variables coincidentes temporalmente, que inicialmente se encontraban

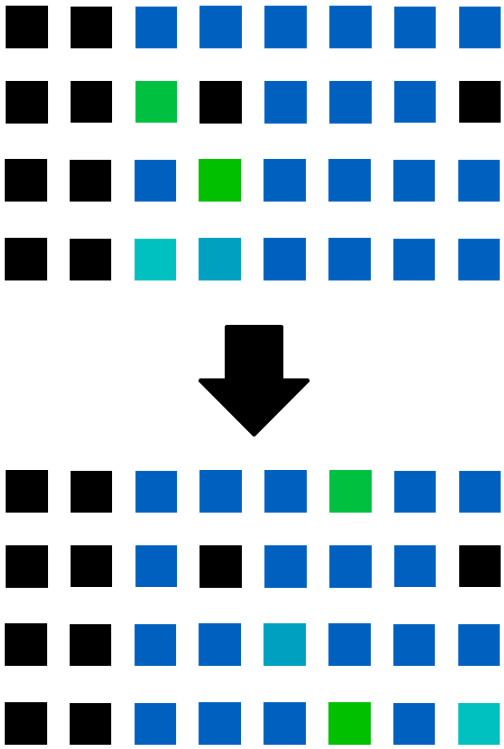
almacenadas juntas en el banco y que han sido separadas por el algoritmo de la siguiente forma:



Veamos, ahora sí, el ejemplo previamente comentado.



Veamos la siguiente figura, para después comentarlas ambas:



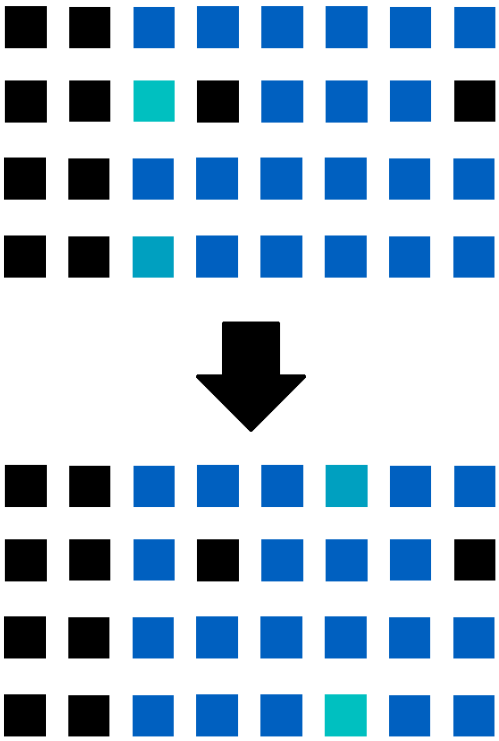
Si comparamos ambar figuras, podemos observar que la variable con mayor tiempo de vida se encuentra almacenada en la parte inferior del banco de registros. En la primera figura, se puede ver cómo el algoritmo ha asignado a una variable con un tiempo de vida reducido un registro cercano. En la siguiente figura, se puede observar el mismo hecho: variables con tiempos de vida reducidos se asignan próximamente al registro al que mayor uso se esté dando, con el fin de que temporalmente el impacto del calor en dicha zona sea el mínimo posible.

Si comparamos las asignaciones del algoritmo con las iniciales en la fase de *register assignment*, observamos cómo la zona inferior izquierda del banco de registros se habría sobrecargado de accesos, con el consiguiente pico de temperatura.

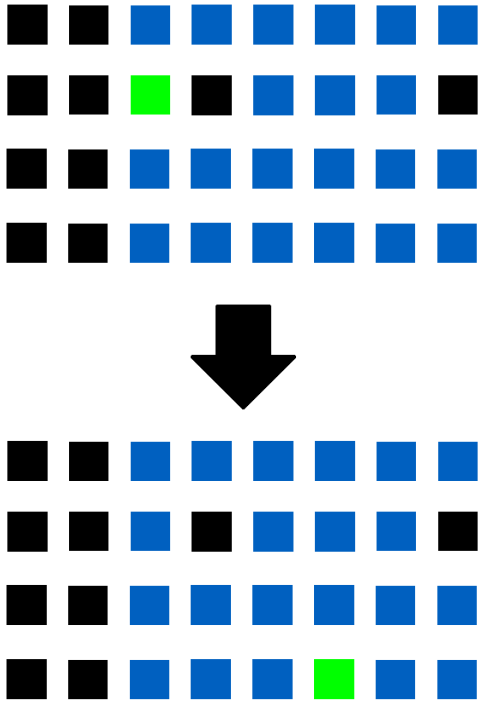
BENCHMARK 5: BUCLE COMPLEJO

El último de los bancos de prueba utilizados ha sido el mismo bucle anteriormente utilizado como banco de pruebas para el primero de los algoritmos.

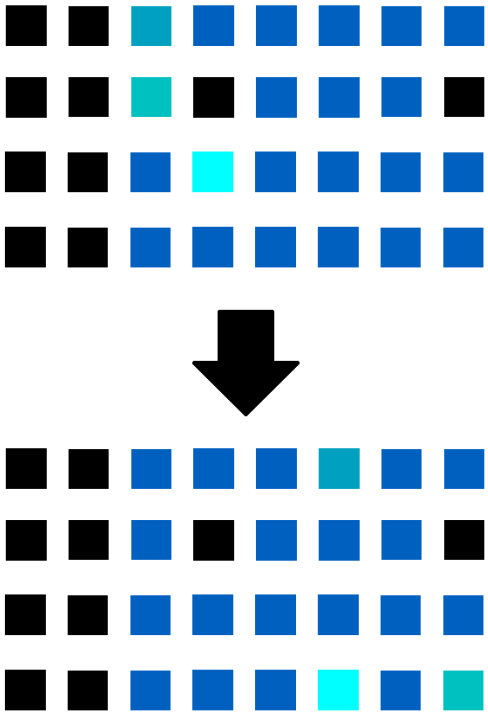
En la primera de las etapas del algoritmo, una de las variables tiene un elevado tiempo de vida, por lo que el algoritmo de nuevo la posiciona en el banco de registros en el mejor de los puntos posibles. Las variables que solapan con ella temporalmente son también colocadas convenientemente:



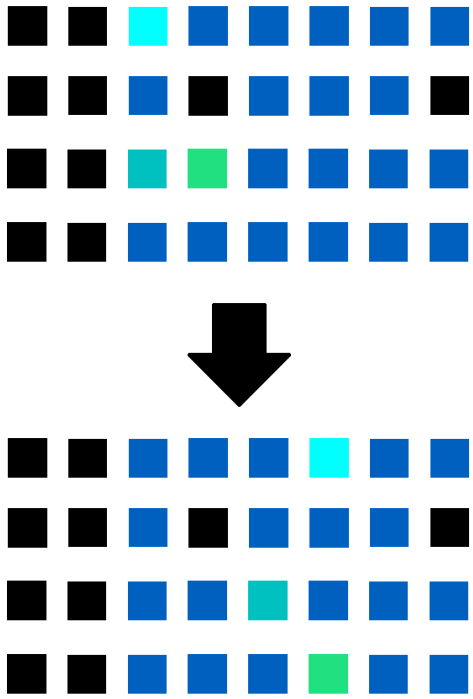
Con el tiempo, observamos que dicha variable vuelve a ser de nuevo aislada por el algoritmo en el banco de registros, hecho que beneficia la disipación de calor en dicho componente:



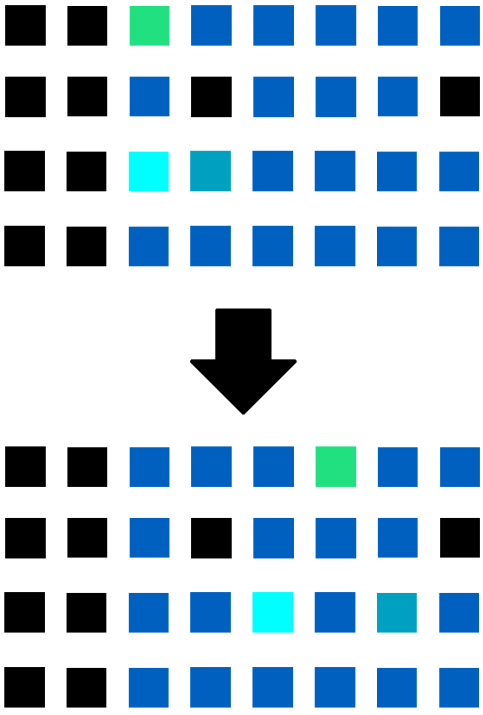
Veremos para finalizar una serie de instantes temporales consecutivos en los que cierta zona del banco de registros se sobrecargará de accesos de lectura y escritura, y cómo el algoritmo lo soluciona:



La figura anterior, como las siguientes, muestran diferentes configuraciones escogidas por el algoritmo para instantes temporales consecutivos, obteniendo una distribución uniforme de los accesos al banco:



En el siguiente instante temporal se obtiene lo siguiente:



Como hemos podido observar en este ejemplo, el algoritmo adapta convenientemente las asignaciones a registros en función de la historia reciente de asignaciones y usos del banco de registros. De ahí que sea un algoritmo tan eficiente en lo relativo a la disipación del calor.

CONCLUSIÓN DEL ESTUDIO DEL ALGORITMO

Con el estudio anterior ha quedado patente la eficiencia del algoritmo y la optimalidad de sus resultados. Además, la carencia que el primero de los algoritmos propuestos sufría ha sido solucionada gracias a que el eje central del algoritmo son los tiempos de vida de las variables del programa.

Así, se ha demostrado que variables con tiempos de vida muy elevados y que solapen temporalmente son alejadas lo máximo posible en el banco de registros, mientras que las variables con tiempos de vida reducidos son posteriormente asignadas con el fin de permitir que en los mejores sitios se asigne a las variables más conflictivas respecto de la temperatura.

También se ha podido observar cómo en el caso de temperaturas excesivamente altas en el banco de registros el algoritmo ha aislado dichos registros. De no aislarse dichos registros se crearían hotspots en el banco de registros, afectando muy seriamente al rendimiento y a la fiabilidad del sistema.

6. RESULTADOS EXPERIMENTALES

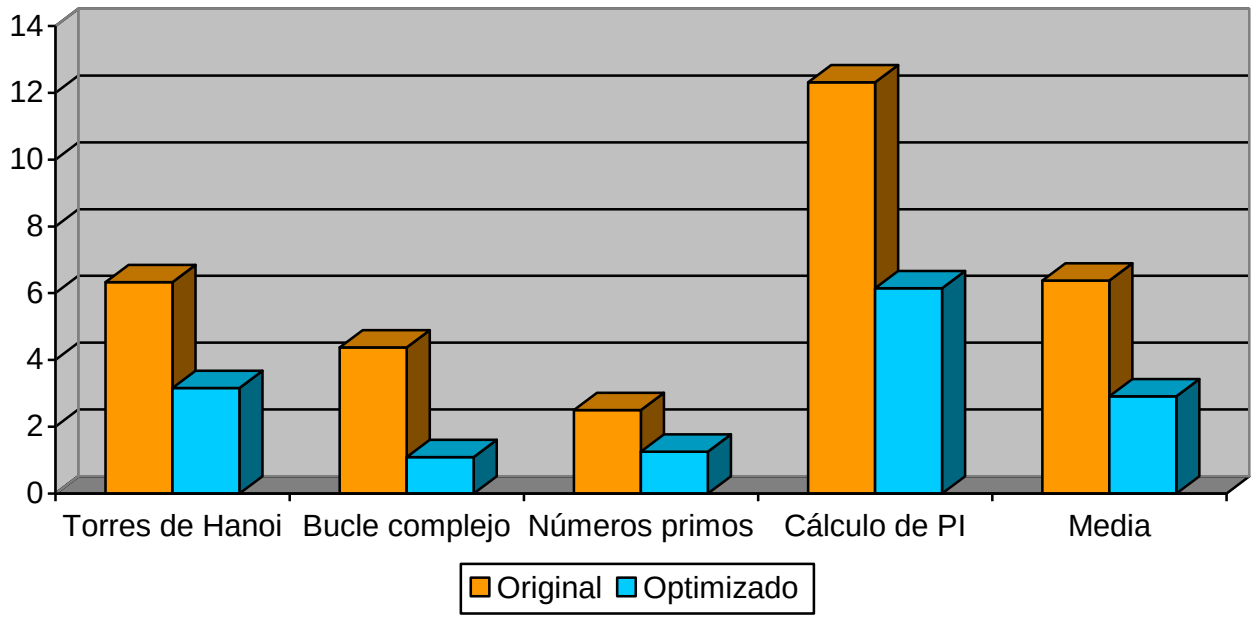
Habiendo aplicado ambos algoritmos propuestos sobre sendos casos de estudio que representan un dominio realista y variado, nos proponemos a comparar los resultados de ambos algoritmos con los obtenidos inicialmente sin aplicar los algoritmos de reasignación, obteniendo de forma conjunta el beneficio obtenido de aplicar cada uno de ellos.

Para el caso del algoritmo de **asignación estática**, el beneficio será proporcional al número de accesos totales a los registros del banco a lo largo de la ejecución de cada programa, e inversamente proporcional al área del chip utilizada por los registros escogidos por el algoritmo. Con ello, calcularemos el **grado de dispersión de la densidad térmica** de cada caso de estudio, mediante la siguiente fórmula:

$$GDDT_1 = \frac{\sum_{i=0}^{regs\_usados} A_i}{Área\_ocupada}$$

donde A_i es el número de accesos totales al registro iésimo durante la ejecución de cada programa, y área_ocupada es máximo de filas por columnas de registros que ocupa cada configuración.

Como podemos observar en la siguiente figura, los beneficios obtenidos al aplicar nuestro primer algoritmo han sido considerables, habiendo conseguido reducir en un **45,62% de media** el grado de dispersión, lo que implica una mayor disipación de calor debido a que se realiza el mismo número de accesos sobre un área mayor de silicio. Los cálculos han sido realizados sobre la configuración de 4 filas y 8 columnas de registros del banco y para la métrica más eficiente (que otorgaba menos peso a la distancia al borde que a la distancia entre registros):

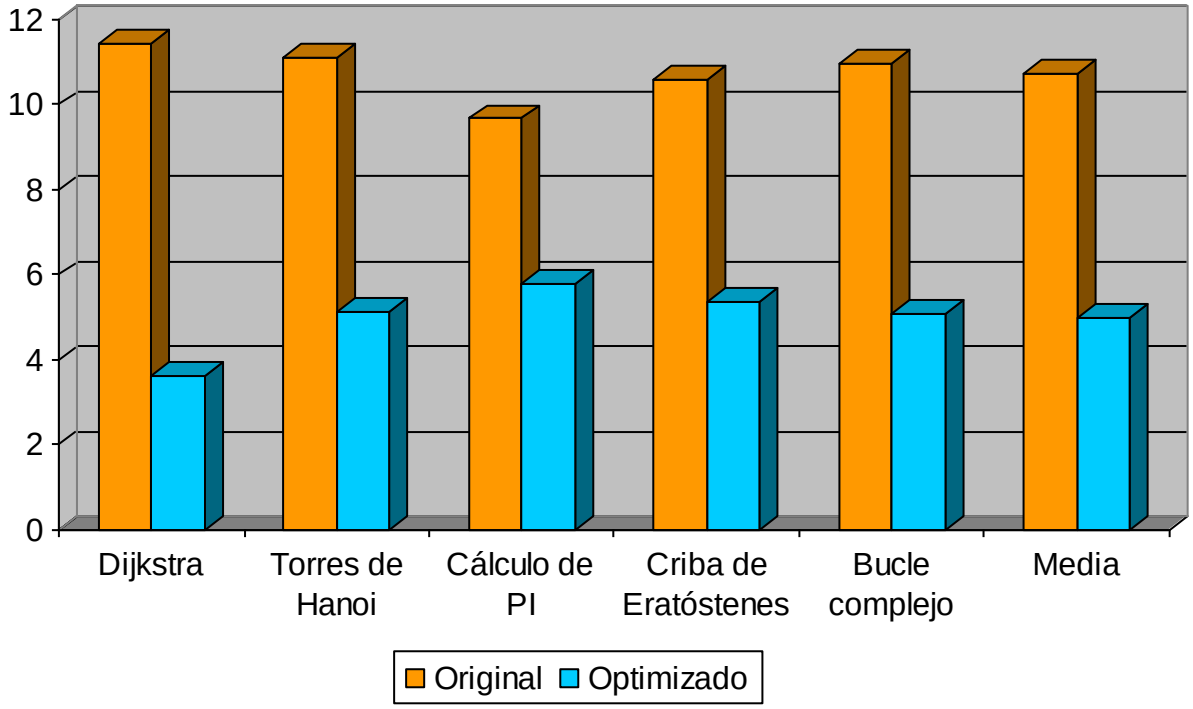


Por otra parte, en el caso del algoritmo de **asignación basada en tiempos de vida** el beneficio ha sido calculado basado en intervalos temporales, al centrarse este algoritmo en los tiempos de vida de las variables. Así, el beneficio en este caso será proporcional al número de accesos que sufre cada registro en un intervalo temporal, mientras que será inversamente proporcional al número de ciclos que posee dicho intervalo. La fórmula que define el **grado de dispersión de la densidad temporal** de cada caso de estudio es la siguiente:

$$GDDT_2 = \frac{\sum_{t=0}^{n} \sum_{i=0}^{regs\_usados} A_i}{t}$$

donde A_i es el número de accesos totales al registro iésimo durante la ejecución del intervalo temporal definido por t, que contiene n instrucciones.

Como podemos observar nuevamente en la siguiente figura los beneficios obtenidos al aplicar nuestro segundo algoritmo han sido considerables, habiendo conseguido reducir en un **46,51% de media** el grado de dispersión:



De nuevo puede observarse una notable mejora con respecto del algoritmo sin optimización. Esta mejora implica una dispersión considerable del número de accesos a cada registro del banco, por lo que generalmente el banco de registro se encontrará a una temperatura más uniforme y los hotspots se habrán eliminado.

7. CONCLUSIONES

Como se ha demostrado a lo largo de este trabajo, la temperatura es un factor muy importante en los microprocesadores. Concretamente, se ha observado que el banco de registros es el componente más problemático, al ser el que mayor cantidad de accesos sufre durante la ejecución de los programas. Por ello, es de suma importancia encontrar políticas y mecanismos que reduzcan los efectos negativos del calor sobre dicho componente.

Así, hemos visto cómo la temperatura en el banco de registros del procesador se puede manejar por el compilador mediante la asignación de los registros físicos. En este trabajo se han propuesto dos políticas de asignación de registros, cuyo objetivo es la optimización térmica del banco de registros: el primero de dichos algoritmos se basa en la dispersión espacial de los registros previamente asignados en la fase de *register assignment,* mientras que el segundo se basa en la dispersión temporal de dichos registros.

Finalmente, se han mostrado resultados experimentales de la aplicación de dichos algoritmos, los cuales muestran la optimización térmica alcanzada.

8. REFERENCIAS

[1] H. Su, F. Liu, A. Devgan, E. Acar, y S. Nassif, “Full leakage estimation considering power supply and temperature variations,” en ISLPED, 2003.

[2] P. Li, L. Pileggi, M. Ashegi, y R. Chandra, “Efficient full-chip thermal modeling and analysis,” en ICCAD, 2004.

[3] W. Huang, E. Humenay, K. Skadron, y M. Stan, “The need for a full-chip and package thermal model for thermally optimized IC designs,” en ISLPED, 2005.

[4] W. Huang, M. Stan, y K. Skadron, “Parameterized physical compact thermal modeling,” IEEE Trans. on Component Packaging and Manufacturing Technology, vol. 28, no. 4, pp. 615–622, December 2005.

[5] S. Lopez-Buedo, J. Garrido, y E. I. Boemo, “Dynamically inserting, operating, and eliminating thermal sensors of FPGA-based systems,” IEEE Trans. on Components and Packaging Technologies, vol. 25, no. 4, pp. 561–566, December 2002.

[6] N. Julien, J. Laurent, E. Senn, y E. Martin, “Power consumption modeling and characterization of the ti c6201,” IEEE Micro, vol. 23, no. 5, pp. 40–49, September 2003.

[7] E. Senn, J. Laurent, N. Julien, y E. Martin, “Softexplorer: Estimation, characterization, and optimization of the power and energy consumption at the algorithmic level,” en International Workshop on Power and Timing Modeling, Optimization and Simulation, 2004.

[8] J. L. Ayala, C. Méndez, y M. López-Vallejo, “Analysis of the Thermal Impact of Source-Code Transformations in Embedded Processors,” en ICECS, 2006.

[9] C. Méndez, J. L. Ayala, y M. López-Vallejo, “Target Independent Thermal Modeling for Embedded Processors,” en IES, 2006.

[10] G. Paci, P. Marchal, F. Polletti, y L. Benini, “Exploring Temperature Aware Design in Low-Power MPSoCs,” en DATE, 2006.

[11] D. Atienza, P. G. D. Valle, G. Paci, y F. Poletti, “A Fast HW/SWFPGA-Based Thermal Emulation Framework for Multi-Processor System-on-Chip,” en DAC, 2006.

[12] Jose L. Ayala, D. Atienza y M. Sabry, “Exploring temeprature-aware Design of memory Architectures in VLIW systems”, en IWIA, 2007 International Workshop on Innovative Architecture for Future Generation Processors and Systems, 2007.

[13] Jose L. Ayala, D. Atienza, “Optimal Loop-Unrolling Mechanisms and Architectural Extensions for an Energy-Efficient Design of Shared Register Files in MPSoCs”, en IEEE Workshop on Innovative Architectures, 2005.

[14] Jose L. Ayala, D. Atienza y M. Sabry, “Energy-aware compilation and hardware design forr VLIW embedded systems”, en IJES: International Journal of Embedded Systems, 2007.

[15] Jose L. Ayala et al. Energy-Aware Compilation and Hardware design for VLIW Embedded Systems. Inderscience International Journal of Embedded Systems, 3(1):73-82, 2007.

[16] Sri Hari Krishna Narayanan et.al Compiler-directed power density reduction in noc-based-core design. En ISQED ’06, pages 570-575, 2006.

[17] James Donald et.al Techniques for multicore thermal managem.nt: Classification and new exploration. En ISCA ’06, pages 78-88, 2006,

[18] Xiangrong Zhou et.al Temperature-aware register reallocation for register le power–density minimization.vACM Trans. Des. Autom. Electron.Sys.,14(2):1-22, 2009.

[19] Benjamin Carrion Schafer et.al Temperature–aware compilation for VLIW processors. En RTCSA ’07, pages 426-431, 2007.

[20] Jose L. Ayala, D. Atienza y M. Sabry, “Thermal-aware compilation for system-on-chip processing architectures”, en ACM Great Lakes Symposium on VLSI, 2010.

[21] (c) Copyright Paul Griffiths 2000, mail@paulgriffiths.net

[22] By Mark Riordan, 24-DEC-1986.

[23] Written by Sanchit Karve